Klaus Jansen
Roberto Solis-Oba (Eds.)

LNCS 2909

Approximation and Online Algorithms

First International Workshop, WAOA 2003
Budapest, Hungary, September 2003
Revised Papers

Springer

Lecture Notes in Computer Science 2909
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Klaus Jansen Roberto Solis-Oba (Eds.)

Approximation and Online Algorithms

First International Workshop, WAOA 2003
Budapest, Hungary, September 16-18, 2003
Revised Papers

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Klaus Jansen
University of Kiel
Institute for Computer Science and Applied Mathematics
Olshausenstr. 40, 24098 Kiel, Germany
E-mail: kj@informatik.uni-kiel.de

Roberto Solis-Oba
University of Western Ontario
Department of Computer Science
London, Ontario, N6A 5B7, Canada
E-mail: solis@csd.uwo.ca

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): F.2.2, G.2.1-2, G.1.2, G.1.6, I.3.5, E.1

ISSN 0302-9743
ISBN 3-540-21079-2 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

© Springer-Verlag Berlin Heidelberg 2004
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 10986639 06/3142 5 4 3 2 1 0

Preface

The Workshop on Approximation and Online Algorithms (WAOA 2003) focused on the design and analysis of algorithms for online and computationally hard problems. Both kinds of problems have a large number of applications arising from a variety of fields. The workshop also covered experimental research on approximation and online algorithms. WAOA 2003 took place in Budapest, Hungary, from September 16 to September 18. The workshop was part of the ALGO 2003 event, which also hosted ESA 2003, WABI 2003, and ATMOS 2003.

Topics of interest for WAOA 2003 were: competitive analysis, inapproximability results, randomization techniques, approximation classes, scheduling, coloring and partitioning, cuts and connectivity, packing and covering, geometric problems, network design, and applications to game theory and financial problems. In response to our call for papers we received 41 submissions. Each submission was reviewed by at least 3 referees, who judged the papers on originality, quality, and consistency with the topics of the conference. Based on these reviews the program committee selected 19 papers for presentation at the workshop and for publication in this proceedings. This volume contains the 19 selected papers and 5 invited abstracts from an ARACNE minisymposium which took place as part of WAOA.

We would like to thank all the authors who responded to the call for papers and the invited speakers who gave talks at the ARACNE minisymposium. We specially thank the local organizer Janos Csirik, the members of the program committee

- Susanne Albers (University of Freiburg)
- Evripidis Bampis (University of Evry)
- Danny Chen (University of Notre Dame)
- Amos Fiat (Tel Aviv University)
- Rudolf Fleischer (Honk Kong University of Science and Technology)
- Pino Persiano (University of Salerno)
- Jose Rolim (University of Geneva)
- Martin Skutella (Max-Planck-Institut für Informatik)

and the subreferees

- Aleksei Fishkin
- Dimitris Fotakis
- Hal Kierstead
- Arie Koster
- Michael Langberg
- Christian Liebchen
- Ulrich Meyer
- Guido Schaefer
- Guochuan Zhang
- Uri Zwick

We gratefully acknowledge sponsorship from the University of Szeged, the EU Thematic Network APPOL (*Approximation and On-line Algorithms*), the EU Research Training Network ARACNE (*Approximation and Randomized Algorithms in Communication Networks*), the DFG Graduiertenkolleg *Effiziente Algorithmen and Mehrskalenmethoden* at the University of Kiel, and the National Sciences and Engineering Research Council of Canada. We also thank Ute Iaquinto, Parvaneh Karimi-Massouleh, Qiang Lu, and Hu Zhang from the research group Theory of Parallelism at the University of Kiel, and Alfred Hofmann and Anna Kramer of Springer-Verlag for supporting our project.

July 2003

Klaus Jansen and Roberto Solis-Oba
Program Chairs

Table of Contents

Contributed Talks

ARACNE Talks

Online Coloring of Intervals with Bandwidth

Udo Adamy[1] and Thomas Erlebach[2,⋆]

[1] Institute for Theoretical Computer Science, ETH Zürich, 8092 Zürich, Switzerland
adamy@inf.ethz.ch
[2] Computer Engineering and Networks Laboratory, ETH Zürich, 8092 Zürich, Switzerland
erlebach@tik.ee.ethz.ch

Abstract. Motivated by resource allocation problems in communication networks, we consider the problem of online interval coloring in the case where the intervals have weights in $(0, 1]$ and the total weight of intersecting intervals with the same color must not exceed 1. We present an online algorithm for this problem that achieves a constant competitive ratio. Our algorithm is a combination of an optimal online algorithm for coloring interval graphs and First-Fit coloring, for which we generalize the analysis of Kierstead to the case of non-unit bandwidth.

1 Introduction

Online coloring of intervals is a classical problem whose investigation has led to a number of interesting insights into the power and limitations of online algorithms. Intervals are presented to the online algorithm in some externally specified order, and the algorithm must assign each interval a color that is different from the colors of all previously presented intervals intersecting the current interval. The goal is to use as few colors as possible. If the maximum clique size of the interval graph is ω, it is clear that ω colors are necessary (and also sufficient in the offline case [6]). Kierstead and Trotter presented an online algorithm using at most $3\,\omega - 2$ colors and showed that this is best possible [11]. Another line of research aimed at analyzing the First-Fit algorithm, i.e., the algorithm assigning each interval the smallest available color. It is known that First-Fit may need at least $4.4\,\omega$ colors [3] on some instances and is therefore not optimal. A linear upper bound of $40\,\omega$ was first presented by Kierstead [8] and later improved to $25.8\,\omega$ by Kierstead and Qin [10].

In this paper, we study a generalization of the online interval coloring problem where each interval has a weight in $(0, 1]$. Motivated by applications, we refer to the weights as bandwidth requirements or simply bandwidths. A set of intervals can be assigned the same color if for any point r on the real line, the sum of the bandwidths of its intervals containing r is at most 1. The special case where every interval has bandwidth 1 corresponds to the original online interval coloring problem. Our problem is thus a simultaneous generalization of online interval coloring and online bin-packing.

Our main result is an online algorithm that achieves a constant competitive ratio for online coloring of intervals with bandwidth requirements. The algorithm partitions

⋆ Partially supported by the Swiss National Science Foundation under Contract No. 21-63563.00 (Project AAPCN) and the EU Thematic Network APPOL II (IST-2001-32007), with funding provided by the Swiss Federal Office for Education and Science.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 1–12, 2004.
© Springer-Verlag Berlin Heidelberg 2004

the intervals into two classes and applies First-Fit to one class and the algorithm by Kierstead and Trotter [11] to the other class. In order to analyze First-Fit in our context, we extend the analysis by Kierstead [8] to the setting where the intervals have arbitrary bandwidth requirements in $(0, 1/2]$.

1.1 Applications

Besides its theoretical interest, investigating the online coloring problem for intervals with bandwidth requirements is motivated by several applications.

First, imagine a communication network with line topology. The bandwidth of each link is partitioned into channels, where each channel has capacity 1. The channels could be different wavelengths in an all-optical WDM (wavelength-division multiplexing) network or different fibers in an optical network supporting SDM (space-division multiplexing), for example. Connection requests with bandwidth requirements arrive online, and each request must be assigned to a channel without exceeding the capacity of the channel on any of the links of the connection. We assume that the network nodes do not support switching of traffic from one channel to another channel (which is the case if only add-drop multiplexers are used). Then a connection request from a to b corresponds to an interval $[a, b)$ with the respective bandwidth requirement, and the problem of minimizing the number of required channels to serve all requests is just our online coloring problem for intervals with bandwidth requirements.

A related scenario in a line network is that the connection requests have unit durations and the goal is to serve all connections in a schedule of minimum duration. In this case, the colors correspond to time slots, and the total number of colors corresponds to the schedule length. This is a special case of the call scheduling problem considered, for example, in [5, 4].

Finally, an interval could represent a time period during which a job must be processed, and the bandwidth of the interval could represent the fraction of a resource (machine) that the job needs during its execution. At any point in time, a machine can execute jobs whose bandwidths (here, resource requirements) sum up to at most 1. If jobs arrive online (before the actual schedule starts) and have to be assigned to a machine immediately, with the goal of using as few machines as possible, we again obtain our online interval coloring problem with bandwidth requirements.

1.2 Related Work

We have already discussed previous work on online coloring of intervals (without bandwidth requirements) in the beginning of the introduction. A survey of results for online graph coloring can be found in [9]. The problem of assigning colors to paths that represent connection requests in communication networks has been studied intensively due to its motivation by all-optical WDM networks. A survey of offline results can be found in [2]. Online path coloring results for trees and meshes are given in [1]. In these path coloring problems, the bandwidth requirement of each path is 1, implying that no two intersecting paths can get the same color.

2 Preliminaries

We are given a collection $\mathcal{I}$ of closed intervals over the real numbers $\mathbf{R}$, where each interval i is associated with a bandwidth requirement $b(i)$, where $0 < b(i) \leq 1$. In an online scenario these intervals are processed one by one in some externally determined sequence. Whenever an interval i arrives, the algorithm knows the previously processed intervals and the colors assigned to them, but it knows nothing about the unprocessed intervals arriving in the future. Based on this knowledge the algorithm irrevocably assigns a color $f(i)$ to the interval i, in such a way that for every color x and every point $r \in \mathbf{R}$ the sum of the bandwidths of the intervals containing r that are assigned color x is at most 1.

We compare the performance of an online algorithm with the number of colors used by an optimal offline algorithm, denoted by OPT. More precisely, we call an online algorithm A c-competitive if $A(\mathcal{I}) \leq c \cdot \mathrm{OPT}(\mathcal{I})$ for all input sequences $\mathcal{I}$.

Let $\mathcal{I} = \{i_1, \ldots, i_\ell\}$ be the collection of intervals and let $b(i_t)$ denote the bandwidth requirement of interval i_t, i.e. $0 < b(i_t) \leq 1$ for $t = 1, \ldots, \ell$. See Fig. 1 for an example with $\ell = 8$ intervals depicted as rectangles. Their projections onto the real line are exactly the intervals over the real numbers, and the height of a rectangle shows the bandwidth requirement of the corresponding interval. Let $(\mathcal{I}, <)$ be the partial order of the intervals in $\mathcal{I}$, where the relation $i < j$ holds if and only if the right endpoint of interval i is less than the left endpoint of interval j, e.g. we have $i_4 < i_5$ in Fig. 1. Let $L \subseteq \mathcal{I}$ be a subset of intervals. For an interval $i \in \mathcal{I}$, we write $i < L$ ($L < i$) if for all intervals $j \in L$, $i < j$ ($j < i$) holds. The *neighborhood* of an interval $i \in \mathcal{I}$ is the set of intervals in $\mathcal{I}$ that are different from i and intersect i. It is denoted by $N(i)$. In the example, the neighborhood of interval i_4 is $N(i_4) = \{i_2, i_3\}$.

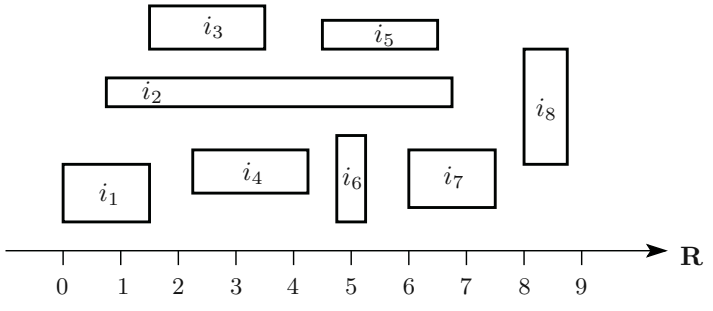


Fig. 1. The set $\mathcal{I} = \{i_1, \ldots, i_8\}$ of intervals.

The *density* $D(r \mid L)$ of L at a point $r \in \mathbf{R}$ is the sum of the bandwidths of the intervals in L that contain the point r, i.e. $D(r \mid L) = \sum_{r \in j \in L} b(j)$. The density $D(i \mid L)$ of an interval $i \in \mathcal{I}$ with respect to L is the minimum density $D(r \mid L)$ over all points r in the interval i, i.e. $D(i \mid L) = \min\{D(r \mid L) : r \in i\}$. With $D(L)$ we denote the maximum density of L, that is $D(L) = \max\{D(r \mid L) : r \in \mathbf{R}\}$. In our example shown in Fig. 1, the density $D(3 \mid \mathcal{I}) = b(i_2) + b(i_3) + b(i_4)$. The density

of the interval i_3 with respect to $\mathcal{I}$ is $D(i_3 \mid \mathcal{I}) = b(i_2) + b(i_3)$ and it is achieved at the point 2 for example. Finally, the maximum density $D(\mathcal{I}) = b(i_2) + b(i_5) + b(i_6)$ is achieved at point 5.

Further, we need some notation for classifying intervals in terms of their left to right order $(\mathcal{I}, <)$. A chain C is a sequence of intervals, where $i < j$ holds for any two consecutive intervals i and j in this sequence C. Therewith, we define the *height* $h(i \mid L)$ of an interval i with respect to L to be the length of the longest chain C in L such that $C < i$. Similarly, the *depth* $d(i \mid L)$ is the length of the longest chain C in L such that $i < C$. The *centrality* of an interval i with respect to L is then given by $c(i \mid L) = \min\{h(i \mid L), d(i \mid L)\}$. Looking again at the example presented in Fig. 1, the height of interval i_4 with respect to $\mathcal{I}$ is $h(i_4 \mid \mathcal{I}) = 1$ since the longest chain consists only of the interval i_1. The corresponding depth $d(i_4 \mid \mathcal{I})$ equals 3 because of the chain $i_6 < i_7 < i_8$ of length 3. Hence, the centrality $c(i_4 \mid \mathcal{I})$ is given by $\min\{1, 3\} = 1$.

Notice that for a subset $L \subseteq N(j)$, if an interval i contains an endpoint of the interval j then $c(i \mid L) = 0$, and if $c(i \mid L) \geq 1$ then the interval i is contained in the interval j, i.e. $i \subset j$.

For our analysis we need two lemmas regarding the existence of an interval with high density in a clique of intervals. The first lemma is a generalization of a lemma that appeared in [12, 7] for the case of interval graphs, i.e. for the setting where all intervals have unit bandwidth.

Lemma 1. *Let $\mathcal{I}$ be a collection of intervals. If L is a clique in the intersection graph of $\mathcal{I}$ (meaning that any two intervals in L intersect), then there exists an interval $i \in L$ such that*

$$D(i \mid L) \geq \frac{D(L)}{2} .$$

Proof. Let $L = \{i_1, \ldots, i_\ell\}$. Since the intervals in L form a clique, there is a point $r \in \mathbf{R}$ that is contained in every interval in L and we have

$$D(L) = \max\{D(r \mid L) : r \in \mathbf{R}\} = \sum_{m=1}^{\ell} b(i_m).$$

We build the sequence S of bandwidths of the intervals in L from left to right. Whenever an interval i_m starts or ends, we append its bandwidth $b(i_m)$ to the sequence S. Thus, the length of the sequence S is 2ℓ since every bandwidth of intervals in L appears twice, once for the left endpoint and once for the right endpoint. Starting from the beginning of S, let S_1 be the largest initial subsequence of elements in S obeying the condition that their sum is less than $D(L)/2$. Likewise, let S_2 be the largest subsequence starting from the end of S (going backwards) under the condition that the sum of elements in S_2 is less than $D(L)/2$.

Now, we claim that there exists an interval i whose bandwidth appears neither in the subsequence S_1 nor in the subsequence S_2. This is the case, because otherwise all bandwidths of intervals would appear either in S_1 or in S_2, which, in turn, means that their total sum would be at least $D(L)$ contradicting the fact that each of the two subsequences sums up to less than $D(L)/2$.

Every such interval i has a density $D(i \mid L) \geq D(L)/2$. This follows from the construction of the sequences S_1 and S_2 and from the fact that in a left-to-right scan through a clique of intervals all left interval endpoints precede all right interval endpoints. □

Lemma 2. *Let L be a clique of intervals each of which is assigned to a different color. If every interval in L has density at least $\rho \leq 1$ within its color class, then there exists an interval $i \in L$ such that the density of i with respect to the union of the present color classes is at least $\rho \cdot |L|/2$.*

Proof. This lemma can be proven similar to Lemma 1 by building a sequence S from a left-to-right scan of the intervals in L, except that this time we append the value ρ whenever an interval in L starts or ends. But we will prove it by induction on the size of L. In the base cases $|L| = 1$ and $|L| = 2$ any interval in L has the desired property, since $|L|/2 \leq 1$. For the induction step, let j be the interval of L with the smallest left endpoint and let k be the interval of L with the largest right endpoint. Using the induction hypothesis, we choose an interval $i \in L \setminus \{j, k\}$ such that the density of the interval i with respect to the union of color classes of intervals in $L \setminus \{j, k\}$ is at least $\rho \cdot (|L|/2 - 1)$. Since the intervals j and k intersect, the interval i is contained in $j \cup k$, and because the intervals j and k have density ρ within their color classes, the density of the interval i with respect to the union of all present color classes is at least $\rho \cdot |L|/2$. □

Finally, if $N = (n_1, \ldots, n_t)$ is a sequence of length t we denote the initial subsequence of N of length k by $N_k = (n_1, \ldots, n_k)$. The result $(n_1, \ldots, n_t, n)$ of appending n to the end of N is denoted by $N \circ n$.

3 The Algorithm

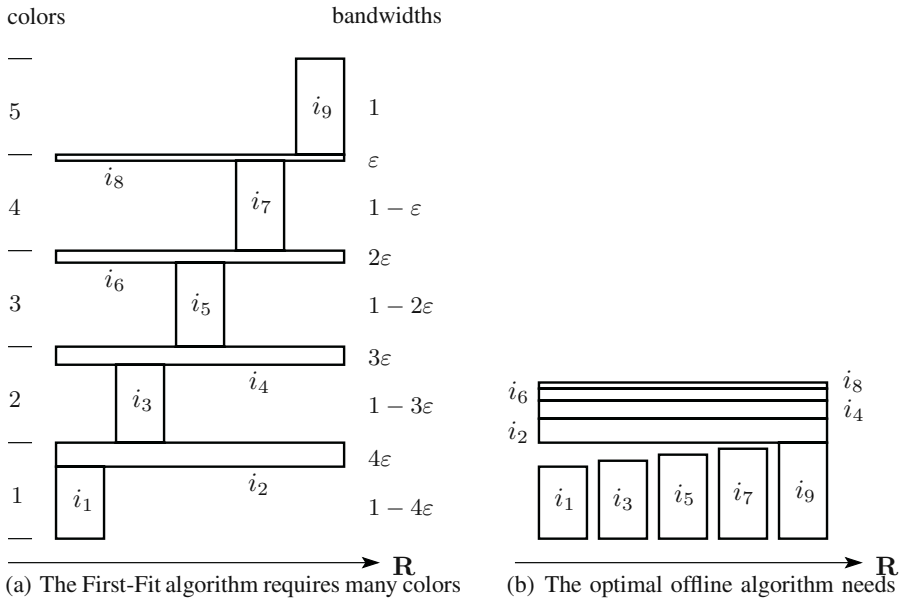
Our algorithm works with two sets C_1 and C_2 of disjoint colors. Let $\sigma = \sigma_1\sigma_2 \cdots$ be any sequence of intervals. Each σ_i is associated with a bandwidth $b(\sigma_i)$, $0 < b(\sigma_i) \leq 1$. Whenever an interval σ_i arrives, we process it according to its bandwidth $b(\sigma_i)$.

If $b(\sigma_i) \leq 1/2$, the algorithm assigns the interval σ_i a color from the set C_1 of colors using the First-Fit principle. The First-Fit assignment is made by coloring the interval σ_i with color α, where α is the smallest color in C_1 such that for every $r \in \sigma_i$, the sum of the bandwidths of intervals previously colored α and containing r is at most $1 - b(\sigma_i)$.

If $b(\sigma_i) > 1/2$, the algorithm assigns the interval σ_i a color from the set C_2 using the optimal online interval coloring algorithm of Kierstead and Trotter [11]. Note that the intervals having bandwidth exceeding $1/2$ have the same property as intervals without bandwidth requirement, namely, if two intervals intersect, they have to receive different colors in a proper coloring.

We remark that the partition into two classes is indeed necessary in our approach, because First-Fit can be arbitrarily bad for intervals with bandwidths in $(0, 1]$ as shown in Fig. 2.

Theorem 1. *The algorithm solves the online problem of coloring intervals with bandwidths using at most* 195 *times as many colors as the optimal offline algorithm.*



(a) The First-Fit algorithm requires many colors for the sequence $i_1, i_2, \ldots, i_9$.

(b) The optimal offline algorithm needs only two colors.

Fig. 2. For bandwidth requirements in $(0, 1]$ the First-Fit algorithm can perform arbitrarily badly.

For the proof of this theorem we need two lemmas which we will prove below. Fix a sequence σ and let $\mathcal{I}$ be the set of all intervals appearing in σ. Let $\mathcal{I}_1 \subset \mathcal{I}$ be the intervals with bandwidth at most $1/2$, and let $\mathcal{I}_2 \subset \mathcal{I}$ be the intervals with bandwidth exceeding $1/2$. We have $\mathcal{I} = \mathcal{I}_1 \dot\cup \mathcal{I}_2$.

Lemma 3. *If the number of colors that the algorithm uses from the set C_1 is at least 2, then this number is less than 192 times the maximum density $D(\mathcal{I}_1)$ of intervals in $\mathcal{I}_1$.*

Lemma 4. *The number of colors from the set C_2 that are used by the algorithm is less than 3 times the number of colors used by an optimal offline algorithm for coloring the intervals in $\mathcal{I}_2$.*

Proof. (of Theorem 1) Let OPT denote the number of colors used by an optimal offline algorithm. Clearly, this number is at least as large as the maximum density of the intervals, i.e. $D(\mathcal{I}) \leq$ OPT. With OPT$(\mathcal{I}_1)$ and OPT$(\mathcal{I}_2)$ we denote the number of colors assigned by an the optimal offline algorithm to the intervals in $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. Since OPT$(\mathcal{I}_1)$ and OPT$(\mathcal{I}_2)$ count the number of colors for restricted sets of intervals, OPT$(\mathcal{I}_1) \leq$ OPT and OPT$(\mathcal{I}_2) \leq$ OPT hold true.

By Lemma 3, the algorithm colors the intervals in $\mathcal{I}_1$ using the First-Fit algorithm with less than $192 \cdot$ OPT$(\mathcal{I}_1)$ many colors. This is true, because OPT$(\mathcal{I}_1) \geq D(\mathcal{I}_1)$ and if the First-Fit algorithm needs just one color for coloring the intervals in $\mathcal{I}_1$, then also the optimal offline algorithm needs one color.

By Lemma 4, the algorithm colors the intervals in $\mathcal{I}_2$ using an optimal online algorithm for coloring interval graphs with less than 3 times as many colors as an optimal offline algorithm.

Thus, the number C of colors used by the algorithm is

$$\begin{aligned} C &< 192 \cdot \mathrm{OPT}(\mathcal{I}_1) + 3 \cdot \mathrm{OPT}(\mathcal{I}_2) \\ &\leq 192 \cdot \mathrm{OPT} + 3 \cdot \mathrm{OPT} \\ &= 195 \cdot \mathrm{OPT}. \end{aligned}$$

Hence, the algorithm is 195-competitive. □

Let us start with the proof of Lemma 4.

Proof. (of Lemma 4) The optimal online interval coloring algorithm uses at most $3 \cdot \omega(\mathcal{I}_2) - 2$ colors for coloring the intervals in $\mathcal{I}_2$ [11]. Since the bandwidth of the intervals in $\mathcal{I}_2$ is greater than $1/2$, any two intersecting intervals must receive different colors. Therefore any offline algorithm needs at least $\omega(\mathcal{I}_2)$ many colors for coloring the intervals in $\mathcal{I}_2$. □

Now we will prove Lemma 3, using a rather technical induction.

Proof. (of Lemma 3)

In this proof we generalize a proof of Kierstead, who proved in [8] that First-Fit uses less than $40\,\omega$ colors for interval coloring, i.e. the case where all bandwidths are 1. We extend the proof to the case where all intervals have bandwidth at most $1/2$.

The main idea of the proof is the following: If the First-Fit coloring uses $x \geq 192 \cdot y$ colors, there is a point $r \in \mathbf{R}$ where the density $D(r \mid \mathcal{I}_1)$ exceeds y. For $y \geq D(\mathcal{I}_1)$ we would arrive at a contradiction, since—by definition of $D(\mathcal{I}_1)$—there will not be any point whose density with respect to $\mathcal{I}_1$ exceeds $D(\mathcal{I}_1)$. Therefore the number of colors used by the First-Fit algorithm is $x < 192 \cdot D(\mathcal{I}_1)$.

Let $f(i)$ denote the color that First-Fit assigns to interval i, and let $C(q) = \{i \in \mathcal{I} : f(i) = q\}$ be the set of intervals with color q. For a range of colors we denote by $C[p, q] = \cup_{p \leq r \leq q} C(r)$ all intervals whose colors are within that range.

In order to find a point with high density in $\mathcal{I}_1$, we construct a sequence I of intervals $i_1 \supset \cdots \supset i_t$ and disjoint blocks $B_1, \ldots, B_t$ of intervals such that $i_m \in B_m$ and the sum of densities $D(i_1 \mid B_1) + \cdots + D(i_m \mid B_m)$ is at least some number s_m, for $m = 1, \ldots, t$. In this construction, every block B_m consists of intervals from the neighborhood of the interval i_{m-1} whose colors are within a certain range. More formally, $B_m = C[x - 192 \cdot r_m + 1, x - 192 \cdot r_{m-1}] \cap N(i_{m-1})$ where $r_m < s_m$, for $m = 1, \ldots, t$. Since the intervals in I form a decreasing sequence, the density of the interval i_t is large, i.e. $D(i_t \mid C[x - 192 \cdot r_t + 1, x]) \geq s_t$. The numbers r_m and s_m will be chosen in such a way that $192 \cdot r_m$ and $192 \cdot s_m$ are integral.

In the induction step we would like to find an interval $i_{t+1} \subset i_t$ with high density in B_{t+1}. However, there may not be any interval with high density in B_{t+1}, or the only intervals with high density in B_{t+1} may overlap i_t. Before we explain below how to deal with these cases, we introduce some more notation.

Let $I = (i_1, \ldots, i_t)$ be a sequence of intervals and let $N = (n_1, \ldots, n_t)$ be a sequence of real numbers, where each n_m is of the form 2^{k-2} for some integer $k \geq 0$.

In particular, we have $n_m \geq 1/4$ for all $m = 1, \ldots, t$. The following parameters are defined on the pair (N, I) for $1 \leq m \leq t$:

- Let s_m be the sum of the first m numbers of the sequence N, i.e. $s_m = \sum_{j=1}^{m} n_j$, and set $s_0 = 0$. Note that $4 \cdot s_m$ is integral, since all n_j's are of the form 2^{k-2}.
- Let $r_m = s_{m-1} + n_m/2 < s_m$, and set $r_0 = 0$. Observe that $8 \cdot r_m$ is integral for the same reason as above.
- Let B_m be the disjoint blocks of intervals defined by $B_m = C[x - 192 \cdot r_m + 1, x - 192 \cdot r_{m-1}] \cap N(i_{m-1})$ for $m \geq 1$, where $i_0 = \mathbf{R}$. B_m consists of all intervals in $\mathcal{I}_1$ that intersect the previous interval i_{m-1} of the sequence I and have a color in the given range.
- Let c_m be the centrality, and d_m be the density of the interval i_m within the intervals in B_m, i.e. $c_m = c(i_m \mid B_m)$ and $d_m = D(i_m \mid B_m)$.

We call a pair (N, I) *admissible* if the following conditions hold for $1 \leq m \leq t$:

(1) The interval i_m is contained in the set B_m, i.e. $i_m \in B_m$.
(2) The centrality of the interval i_m with respect to B_m is at least 2, i.e. $c_m \geq 2$, if $m > 1$.
(3) The number $n_{m+1} \geq n_m 4^{2-c_m+1}$, if $m < t$.
(4) The density of the interval i_m with respect to B_m is at least n_m, i.e. $d_m \geq n_m$.

Notice that if (N, I) is an admissible pair, then the density of the interval i_t is at least s_t, i.e. $D(i_t \mid \mathcal{I}_1) \geq s_t$. Since the blocks B_m are disjoint, and the intervals i_m are contained in each other by (2), the density of the interval i_t is at least the sum of the densities d_m, which in turn are at least n_m by (4). By definition s_t constitutes exactly this sum.

We shall prove the lemma by first showing that there exists an admissible pair (N, I), and if (N, I) is such an admissible pair and the number x of used colors fulfills $x \geq 192 \cdot s_t$ then there exists an admissible pair (N', I') such that $s'_{t'} > s_t$. Here and below primes indicate elements and parameters of the new admissible pair.

For the basis of the induction we need an admissible pair. Let i be any interval colored x.

If $b(i) = 1/2$, then $N = (1/2)$ and $I = (i)$ is an admissible pair consisting of $t = 1$ interval. Its parameters are as follows: n_1 and therefore s_1 equal $1/2$, and $r_1 = n_1/2 = 1/4$. Since the interval $i_1 = i$ is colored x, it is contained in the block $B_1 = C[x - 48 + 1, x]$. Further its density d_1 in B_1 is at least $1/2 = n_1$, since the interval i_1 itself contributes a bandwidth $b(i_1) = 1/2$. The conditions (2) and (3) do not play a role for $t = 1$.

If $b(i) < 1/2$, then because of First-Fit there is a point $r \in i$, where intervals colored $x-1$ intersect, whose total bandwidth exceeds $1/2$. Otherwise the interval i would have been colored $x - 1$. These intervals form a clique L and according to Lemma 1, there exists an interval $j \in L$ with density $D(j \mid L) > 1/4$. Then $I = (j)$ and $N = (1/4)$ is an admissible pair. Again we have to check the conditions. The numbers n_1 and s_1 are $1/4$, whereas $r_1 = n_1/2 = 1/8$. Since the interval $i_1 = j$ is colored $x - 1$, it is contained in the block $B_1 = C[x - 24 + 1, x]$. The condition (4) is fulfilled, because the density of the interval is $d_1 > 1/4 = n_1$.

Now fix an admissible pair (N, I). We begin the process of constructing (N', I'). Let $B = C[x - 192 \cdot s_t + 1, x - 192 \cdot r_t] \cap N(i_t)$.

The bandwidth of the interval i_t is $b(i_t) \leq 1/2$. Since $i_t \in B_t$, the color $f(i_t)$ of the interval i_t is larger than $x - 192 \cdot r_t$. According to First-Fit, there exists for each color $\alpha \in [x - 192 \cdot s_t + 1, x - 192 \cdot r_t]$ a point $r_\alpha \in i_t$ such that there are intervals with color α intersecting at r_α, whose bandwidths sum up to more than $1/2$. Otherwise the interval i_t would have received a smaller color. Since $s_t - r_t = n_t/2$, there are at least $96 \cdot n_t$ places r_α where intervals with total bandwidth greater than $1/2$ intersect, and according to Lemma 1, we find at least $96 \cdot n_t$ intervals of different colors each with density exceeding $1/4$ within its color class. Let M denote this set of intervals.

We partition M into level sets according to the left-to-right ordering of the intervals. Let $L_k = \{i \in M : h(i \mid M) = k \leq d(i \mid M)\}$ and $R_k = \{i \in M : h(i \mid M) > k = d(i \mid M)\}$. For an interval $i \in M$, $i \in L_k \cup R_k$ or $i \in M \setminus \bigcup_{j<k}(L_j \cup R_j)$ implies that the centrality $c(i \mid M)$ of i with respect to M is at least k.

By the pigeon hole principle (at least) one of the following four cases holds true. Otherwise M would contain fewer than $96 \cdot n_t$ intervals.

(i) There is an interval $i \in M$ with high centrality $c(i \mid M) \geq 3 + \lceil \log_4 n_t \rceil$.
(ii) There is a large central level set, i.e. $|L_k| \geq 8n_t 4^{2-k}$ or $|R_k| \geq 8n_t 4^{2-k}$ for some k such that $2 \leq k \leq 2 + \lceil \log_4 n_t \rceil$.
(iii) There is a large first level set, i.e. $|L_1| \geq 8n_t$ or $|R_1| \geq 8n_t$.
(iv) There is an extra large outer level set, i.e. $|L_0| \geq 28n_t$ or $|R_0| \geq 28n_t$.

– Case (i):
If there is an interval $i \in M$ with high centrality, we extend the sequence I by this interval i. The length of the sequence increases by 1. Hence, we set $t' = t + 1$, $i'_{t+1} = i$, and $n'_{t+1} = 1/4$, because the interval i'_{t+1} has density greater than $1/4$ within its color class.
We have to show that $N' = N \circ n'_{t+1}$ and $I' = I \circ i'_{t+1}$ form an admissible pair (N', I'). For condition (1) we have to observe that the interval i'_{t+1} is chosen from B which is a subset of B'_{t+1}, because $r'_{t+1} = s_t + n'_{t+1}/2 > s_t$. Hence, $i'_{t+1} \in B'_{t+1}$. Condition (2) holds, because $c'_{t+1} \geq 3 + \lceil \log_4 n_t \rceil \geq 2$ since $n_t \geq 1/4$. The number $n'_{t+1} = 1/4 = 4^{-1} = n_t 4^{2-3-\log_4 n} \geq n_t 4^{2-c'_{+1}}$ fulfills the condition (3), and the density $d'_{t+1} \geq 1/4 = n'_{t+1}$, which establishes condition (4). The density of the interval $i'_{t'}$ with respect to $\mathcal{I}_1$ is at least $s'_{t'} = s_t + 1/4$.

– Case (ii):
If there is a large central set, we can without loss of generality assume that $|L_k| \geq 8n_t 4^{2-k}$. The other case is exactly symmetric. Note that L_k is a clique in $\mathcal{I}_1$ since $i, j \in M$ and $i < j$ implies that $h(i \mid M) < h(j \mid M)$. Furthermore, each interval in L_k has a density greater $1/4$ within its color. Thus, by applying Lemma 2 with $\rho = 1/4$ we can find an interval $i \in L_k$ with density $D(i \mid B) > n_t 4^{2-k}$. We extend the sequence I by this interval i and set the values $t' = t + 1$, $i'_{t+1} = i$, and $n'_{t+1} = n_t 4^{2-k}$. Note that $n'_{t+1} \geq 1/4$ since $n'_{t+1} = n_t 4^{2-k} \geq n_t 4^{2-2-\lceil \log_4 n \rceil} \geq n_t 4^{-\log_4 n - 1} = 1/4$.

Again, we have to verify that $N' = N \circ n'_{t+1}$ and $I' = I \circ i'_{t+1}$ form an admissible pair (N', I'). The condition (1) is satisfied, because the interval i'_{t+1} is chosen from B, which is a subset of B'_{t+1}, since $r'_{t+1} = s_t + n'_{t+1}/2 > s_t$. The centrality $c'_{t+1} = k \geq 2$ and the number $n'_{t+1} = n_t 4^{2-k}$ fulfill the conditions (2) and (3). The condition (4) concerning the density of the interval i'_{t+1} holds, because $d'_{t+1} > n_t 4^{2-k} = n'_{t+1}$.

Hence, the density of the interval $i'_{t'}$ with respect to $\mathcal{I}_1$ is at least $s'_{t'} = s_t + n_t 4^{2-k}$.

– Case (iii)

If there is a large first level set, we can without loss of generality assume that $|L_1| \geq 8n_t$. For the same reason as in Case (ii) the set L_1 is a clique in $\mathcal{I}_1$, and each interval in L_1 has a density greater $1/4$ within its color class. Hence, by Lemma 2 (again with $\rho = 1/4$) there exists an interval $i \in L_1$ with density $D(i \mid B) > n_t$.

Since this interval i with high density in B'_{t+1} might have a centrality of one only, thus violating the condition (2), we replace the interval i_t with the interval i. We set $t' = t$, $i'_t = i$, and $n'_t = 2n_t$.

Then the new block $B'_t \supseteq B_t \cup B$, since $r'_t = s_{t-1} + n'_t/2 = s_t$. Also the interval $i'_t \subset i_t$, because i'_t has centrality 1 in the subset M of the neighborhood $N(i_t)$. Hence, $i'_t \in B'_t$ establishing the condition (1). The condition (2) simply holds, because we have $c'_t \geq c_t \geq 2$, if $t > 1$. For condition (3) a rough estimate suffices, namely $n'_t > n_t \geq n_{t-1} 4^{2-c} \geq n'_{t-1} 4^{2-c'}$. The density condition (4) is satisfied, because $d'_t \geq D(i \mid B) + d_t > 2n_t = n'_t$.

In total, the density of the interval $i'_{t'}$ with respect to $\mathcal{I}_1$ is at least $s'_{t'} = s_t + n_t$.

– Case (iv)

If there is an extra large outer level set, we can without loss of generality assume that $|L_0| \geq 28n_t$. Each interval in L_0 has density greater $1/4$ within its color class. Let e be the leftmost point that is contained in the interval i_t as well as in all intervals in L_0. We define $T := B_t \cup B$. Then, the density $D(e \mid T)$ of the point e with respect to T is at least the density of the interval i_t with respect to B_t plus the density of e within the set L_0. Hence, $D(e \mid T) \geq D(i_t \mid B_t) + D(e \mid L_0) > n_t + 7n_t = 8n_t$, and according to Lemma 1 we find an interval j_1 in the clique of intervals in T containing e, such that $D(j_1 \mid T) \geq 4n_t$.

Additionally, we choose an interval j_2 from B_t such that its centrality $c(j_2 \mid B_t) = c_t - 1$ and $j_2 < i_t$. Note that such an interval must exist, because of the centrality of the interval i_t in B_t.

Now we distinguish two cases according to the left interval endpoints of j_1 and j_2. Either the left endpoint of j_2 lies further left, which implies that the centrality $c(j_1 \mid T) \geq c(j_2 \mid T) \geq c_t - 1$, or the left endpoint of the interval j_1 lies further left, in which case the interval j_2 is contained in j_1, and thus the density $D(j_2 \mid T) \geq D(j_1 \mid T) \geq 4n_t$.

In either case there exists an interval $i \in \{j_1, j_2\}$ in T such that $D(i \mid T) \geq 4n_t$ and $c(i \mid T) \geq c_t - 1$. In order to fulfill the condition (2) we have to replace either the interval i_t or the interval i_{t-1} with the new interval i depending on the centrality $c(i \mid T)$.

- If the centrality $c(i \mid T) \geq 2$ or $t = 1$, we replace the interval i_t with the new interval i, and set the values $t' = t$, $i'_t = i$, and $n'_t = 4n_t$.
  The new block $B'_t \supset T$, since $r'_t = s_{t-1} + n'_t/2 = s_{t-1} + 2n_t = s_t + n_t > s_t$. Because of the centrality of the interval i'_t the condition (2) is clearly satisfied, and we have also that $i'_t \subset i_{t-1}$, if $t > 1$. Hence, the interval i'_t is contained in B'_t fulfilling the condition (1). To verify the condition (3) we need a short calculation. We have $n'_t = 4n_t \geq 4n_{t-1}4^{2-c} \geq n'_{t-1}4^{2-c'}$, where the last inequality holds because $c'_t \geq c_t - 1$. The condition (4) holds because the density $d'_t \geq 4n_t = n'_t$.
  Hence, the density of the interval $i'_{t'}$ with respect to $\mathcal{I}_1$ is at least $s'_{t'} = s_t + 3n_t$.
- Otherwise $c(i \mid T) = 1$ and $t > 1$. Thus the centrality $c_t = 2$ and $n_t \geq n_{t-1}$. In this case we discard the interval i_t and replace the interval i_{t-1} with the new interval i. We set the numbers $t' = t - 1$, $i'_{t-1} = i$ and $n'_{t-1} = 4n_t$.
  We are left to show that the sequences $N' = N_{t-2} \circ n'_{t-1}$ and $I' = I_{t-2} \circ i'_{t-1}$ form an admissible pair (N', I').
  For condition (1) observe that the new block $B'_{t-1} \supseteq B_{t-1} \cup T$, since $r'_{t-1} = s_{t-2} + n'_{t-1}/2 = s_{t-2} + 2n_t \geq s_{t-2} + n_{t-1} + n_t = s_t$. Because of the centrality of the interval i'_{t-1} we have $i_{t-1} \supset i'_{t-1}$. Thus, we can conclude that $i'_{t-1} \in B'_{t-1}$. Since the interval i'_{t-1} is contained in the interval i_{t-1}, the centrality $c'_{t-1} \geq c_{t-1}$, which establishes the condition (2), because $c_{t-1} \geq 2$, if $t > 2$. The condition (3) holds, since $n'_{t-1} = 4n_t > n_{t-1} \geq n_{t-2}4^{2-c\ -1} \geq n'_{t-2}4^{2-c'-1}$. Finally, the density condition (4) is fulfilled because $d'_{t-1} \geq D(i \mid T) + d_{t-1} \geq 4n_t + n_{t-1} > 4n_t = n'_{t-1}$.
  This time, the density of the interval $i'_{t'}$ with respect to $\mathcal{I}_1$ is at least $s'_{t'} = s_{t-2} + 4n_t \geq s_{t-2} + n_{t-1} + n_t + 2n_t = s_t + 2n_t$. □

4 Conclusion

We have presented an online algorithm for coloring intervals with bandwidth requirements that achieves a constant competitive ratio, where the constant is at most 195. This large constant is presumably not optimal, since already the analysis of the First-Fit algorithm for interval graphs by Kierstead [8] is not optimal. We extend this analysis to the case of bandwidth requirements in $(0, 1/2]$. At this time we do not know how to generalize the optimal online algorithm for interval graphs [11] to the case of arbitrary bandwidths between 0 and $1/2$. An improved algorithm or a better analysis of the First-Fit method would be interesting future research.

References

1. Y. Bartal and S. Leonardi. On-line routing in all-optical networks. *Theoretical Computer Science*, 221(1–2):19–39, 1999.
2. B. Beauquier, J.-C. Bermond, L. Gargano, P. Hell, S. Perennes, and U. Vaccaro. Graph problems arising from wavelength-routing in all-optical networks. In *Proceedings of the Second Workshop on Optics and Computer Science (WOCS), part of IPPS'97*, 1997.

3. M. Chrobak and M. Ślusarek. On some packing problems related to dynamic storage allocation. *RAIRO Informatique Théoretique et Applications*, 22:487–499, 1988.
4. T. Erlebach and K. Jansen. Off-line and on-line call-scheduling in stars and trees. In *Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science*, LNCS 1335, pages 199–213, 1997.
5. A. Feldmann. On-line call admission for high-speed networks (Ph.D. Thesis). Technical Report CMU-CS-95-201, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1995.
6. M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.
7. A. Gyárfás and J. Lehel. Online and First-Fit colorings of graphs. *Journal of Graph Theory*, 12:217–227, 1988.
8. H. A. Kierstead. The linearity of First-Fit coloring of interval graphs. *SIAM Journal on Discrete Mathematics*, 1(4):526–530, 1988.
9. H. A. Kierstead. Coloring graphs on-line. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, LNCS 1442. Springer-Verlag, Berlin, 1998.
10. H. A. Kierstead and J. Qin. Coloring interval graphs with First-Fit. *SIAM Journal on Discrete Mathematics*, 8:47–57, 1995.
11. H. A. Kierstead and W. T. Trotter, Jr. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.
12. H. S. Witsenhausen. On Woodall's interval problem. *Journal of Combinatorial Theory (Series A)*, 21:222–229, 1976.

Open Block Scheduling in Optical Communication Networks*

Alexander A. Ageev[1,**], Aleksei V. Fishkin[2,***], Alexander V. Kononov[1,3], and Sergey V. Sevastianov[1,†]

[1] Sobolev Institute of Mathematics
pr. Koptyuga 4, 630090 Novosibirsk, Russia
{ageev,alvenko,seva}@math.nsc.ru
[2] Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, 24 098 Kiel, Germany
avf@informatik.uni-kiel.de
[3] National Chi Nan University
1 University Rd., Puli, Nantou, Taiwan, 545 R.O.C

Abstract. In this paper the process of data transmission in optical communication networks is modeled as a shop-type scheduling problem, where channels (wavelengths) are treated as machines. We formulate an *Open Block problem* with the minimum makespan objective (an $OB||C_{\max}$ problem) in which a relation of a new type between the operations of each job is introduced: any two operations of a job have identical processing times and may be processed either completely simultaneously (in a common *block*) or, alternatively, with full diversity in time. We show that the problem is polynomially solvable for 4 machines, binary NP-hard for 6 machines and strongly NP-hard for a variable number of machines. Adding release dates to the two-machine problem also leads to the NP-hardness in strong sense. For the case of a variable number of machines we present a polynomial time $\sqrt{2}$-approximation algorithm and prove that there is no polynomial time ρ-approximation algorithm with $\rho < 11/10$, unless P=NP. For the case when the number of machines is fixed, we show that the problem can be solved by a linear time PTAS and by a few linear time statistically optimal algorithms (generating optimal schedules for almost all instances).

1 Introduction

In this paper we study a scheduling problem that models the process of data transmission in optical communication networks with n nodes and m channels.

* This is a reduced version of a full paper to be submitted to a journal.

** Supported by the Russian Foundation for Basic Research, project codes 01-01-00786, 02-01-01153 and by INTAS, project code 00-217, and by the Programme "Universities of Russia", project code UR.04.01.012.

*** Supported by EU-Project CRESCO, IST-2001-33135.

† Supported by the Russian Foundation for Basic Research, project code 02-01-00039.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 13–26, 2004.
© Springer-Verlag Berlin Heidelberg 2004

We assume that each node has a receiver which is pre-tuned to a fixed channel, and has a "tunable" transmitter (laser) which can transmit on several channels simultaneously. Each node on the transmitting side has a multicast data packet that must be delivered to all receiving nodes in its destination set. The transmission of a data packet completes when each destination node receives at least one copy of the packet. The goal is to minimize the overall transmission time, i.e., the time for all data packets to be received. We consider a multicast service with *fanout splitting*, which means that any data packet can be split into several packets, identical by their contents but having element-wise different sets of destination channels. (For details see [8], [14].)

Data transfer in such optical networks must meet the following natural restrictions:
(a) at any point in time no two transmitters may transmit information on the same wavelength (channel);
(b) at most one data packet can be sent by a transmitter at a time (although, on several channels simultaneously). Below we formulate a new scheduling problem based on these conditions.

Open Block Problem (short notation: $OB||C_{\max}$). We are given a set of n jobs $\boldsymbol{J} = \{J_1, \ldots, J_n\}$ and a set of m machines $\boldsymbol{M} = \{M_1, \ldots, M_m\}$. For each job J_j, a processing time p_j and a subset of indices $v_j \subseteq \{1, \ldots, m\}$ are specified, so that job J_j consists of $|v_j|$ operations that have to be processed on machines $\{M_i \,|\, i \in v_j\}$ and have identical processing times equal to p_j. Any two operations of a job may run either simultaneously (in a *block*), or should not overlap in time. No restrictions on the combining of operations into blocks and on the order of processing the blocks are specified. Each machine can process at most one operation at a time. Preemption is not allowed. The goal is minimizing the makespan. (Fig. 1 illustrates a feasible schedule for a given instance of the OB-problem.)

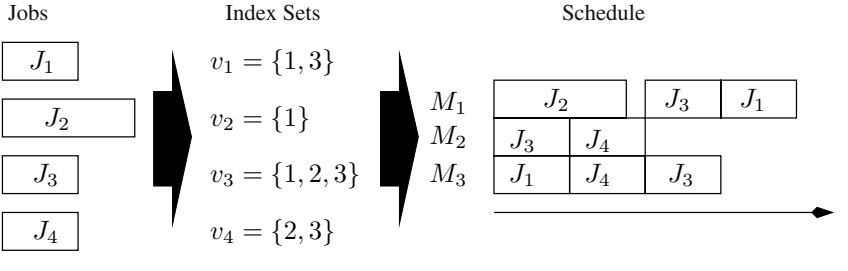


Fig. 1. A feasible open-block schedule for jobs $J_1, \ldots, J_4$ on three machines

Thus, a novel feature of the above problem is a relation of a new type between the operations: now any two operations of a job may be processed either completely simultaneously, or with a full diversity in time.

Related Problems and Results. Clearly, given an instance of the OB-problem, it can also be treated as an instance of the *Multiprocessor Tasks* problem (MPT) — on one hand, and of the *Open Shop* problem — on the other hand. Moreover, any schedule constructed for that instance and feasible with respect to either the MPT, or the Open Shop problem is also feasible with respect to the OB-problem, and therefore, efficient methods elaborated for those two problems can be useful to provide good solutions for the OB-problem as well.

Let us first review the known results for the $O||C_{\max}$ problem. As is known from [6], the $O2||C_{\max}$ problem admits a linear-time solution, while the $O3|p_{ij} = p_j|C_{\max}$ problem is binary NP-hard. NP-hardness of the general $O||C_{\max}$ problem in strong sense follows from the result proved in [15]: deciding whether there exists a schedule of length at most 4 is strongly NP-complete. This also implies that for the $O||C_{\max}$ problem with a variable number of machines and any $\rho < 5/4$ there is no ρ-approximation algorithm, unless $\mathbb{P} = \mathbb{NP}$. At the same time, it is well known that any greedy algorithm provides a 2-approximation. For the $Om||C_{\max}$ problem (when the number of machines m is arbitrary but fixed), we can suggest a polynomial time approximation scheme (PTAS) [12]. Efficient statistically optimal algorithms able to construct optimal schedules for almost all instances of the $O||C_{\max}$ problem were designed in [3, 9–11].

The MPT-problem seems to be much harder. Already $MPT3||C_{\max}$ is strongly NP-hard [7]. However, $MPTm|p_j = 1|C_{\max}$ can be solvable in polynomial time [7]. A PTAS is proposed for $MPTm||C_{\max}$ in [1]. On the other hand, $MPT|p_j = 1|C_{\max}$ cannot be approximated within a factor of $m^{1/2-\varepsilon}$, unless $\mathbb{P} = \mathbb{NP}$ [4]. Matching this, polynomial time $O(\sqrt{m})$-approximation algorithms for $MPT|p_j = 1|C_{\max}$ and $MPT||C_{\max}$ were recently proposed in [2].

Our Results. It is shown that the problem can be solved in polynomial time, if the number of machines is at most 4, whereas the 6-machine problem is binary NP-hard. The problem with a variable number of machines is shown to be strongly NP-hard, while adding different release times makes the problem strongly NP-hard already for two machines. The problem with a variable number of machines admits a polynomial time $\sqrt{2}$-approximation, but cannot be approximated better than within a factor of 11/10 of the optimum, unless $\mathbb{P} = \mathbb{NP}$. In contrast to the above result, for the case of an arbitrary fixed number of machines it can be easily shown that the problem admits a linear time PTAS.

As an easy corollary of results known for the Open Shop problem, we also establish that the problem becomes polynomially solvable for the set of instances with sufficiently large values of the maximum machine load — in comparison with the maximum job processing time. (It can be shown that on the set of instances with a fixed number of machines this property almost always holds for increasing number of jobs.)

The paper is organized as follows. In Section 2 we introduce necessary notions and notation. In Section 3 a few known polynomially solvable cases of the OB-problem are presented. Section 4 contains NP-hardness and inapproximability results. A few constant-factor approximation algorithms and a PTAS are presented in Section 5.

2 Preliminaries

We define the *maximum processing time* of an operation as $p_{\max} = \max_j p_j$, and the *length* d_j *of job* $J_j \in \boldsymbol{J}$ as the total length of all its operations; $d_{\max} = \max_{J \in \boldsymbol{J}} d_j$ stands for the *maximum job length* over all jobs. The total processing time of the operations on machine M_i is denoted by ℓ_i and called the *load* of machine M_i; $\ell_{\max} = \max_i \ell_i$ and $C_{\max}(S)$ denote the *maximum machine load* and the *length of schedule* S respectively; OPT stands for the optimum.

We say that a polynomial-time algorithm A is a ρ-*approximation* algorithm (or provides a ρ-*approximation*) for a minimization problem, if for any instance of the problem it outputs a feasible solution of cost at most $\rho \cdot \mathrm{OPT}$. We say that a minimization problem admits a polynomial-time approximation scheme (PTAS) if it can be solved by a polynomial-time $(1+\varepsilon)$-approximation algorithm for any fixed $\varepsilon > 0$.

It is clear that for any instance of the OB-problem (as well as of any scheduling problem on dedicated machines with the standard constraint that each machine cannot process more than one operation at a time),

$$\ell_{\max} \leq \mathrm{OPT}. \tag{1}$$

Normal Schedules and Instances. Given an instance I of a scheduling problem, its feasible schedule S is called *normal*, if

$$C_{\max}(S) = \ell_{\max}. \tag{2}$$

Due to (1), any normal schedule is optimal. The converse is not true, in general. An instance I is called *normal*, if its optimal schedule is normal.

On the face of it, such a coincidence like in (2) seems to be improbable. However, it will be shown in section 3 that most instances of the OB-problem are normal, which approves the term.

No-Idle Schedules. We say that a schedule is *left no-idle* (*right no-idle*), or *LNI* (*RNI*), for short, if each machine $M_i \in \boldsymbol{M}$ executes all its operations within a connected time interval $I_i = [t'_i, t''_i]$ (i.e., without inner idle time), and all intervals I_i start (finish) at the same point in time $t'_i = t$ ($t''_i = t$).

Singles and Multi-jobs. A job J_j is called a *single*, if it consists of only one operation. Otherwise, it is called a *multi-job*.

3 Polynomially Solvable Cases

Theorem 1. *Every instance of the OB-problem with 4 machines and n jobs has a normal schedule which can be found in* $O(n)$ *time.*

Proof (sketch). We proceed as follows. First, we examine the list of jobs, $J_1, J_2, \ldots, J_n$, and glue any two jobs J_j and J_k together each time, when their index sets coincide ($v_j = v_k$), i.e., they have to be executed on the same sets of

machines. We terminate gluing as soon as all jobs have distinct index sets. This requires at most $O(n)$ time.

We obtain an instance containing 4 *singles*, 6 *double-jobs*, 4 *triple-jobs*, and a single multi-job with 4 operations. The latter can be removed from our instance without loss of generality: after constructing a normal schedule for the remaining instance, we can simply add the removed job as a single block at the end of the schedule. So, let now $\ell_{\max}$ stand for the maximum machine load in the remaining instance I^*.

Next, we aim at constructing a normal schedule. Let J_i $(i = 1, \ldots, 4)$ stand for the triple-job with the missed operation on machine M_i, and let J_{ij} denote the double-job with two operations on machines M_i and M_j; p_i and p_{ij} will denote the processing times of those jobs, respectively. We define three numbers:

$$\alpha_{12} = \min\{p_{12}, p_{34}\},\ \alpha_{13} = \min\{p_{13}, p_{24}\},\ \alpha_{14} = \min\{p_{14}, p_{23}\},$$

and compute $\alpha^* = \max\{\alpha_{12}, \alpha_{13}, \alpha_{14}\}$. Assuming w.l.o.g. that $\alpha^* = \alpha_{12}$, we divide the set of all multi-jobs into two subsets: $\boldsymbol{J'} = \{J_{12}, J_{13}, J_{24}, J_3, J_4\}$ and $\boldsymbol{J''} = \{J_{34}, J_{14}, J_{23}, J_1, J_2\}$. We construct a feasible LNI-schedule starting at time 0 for the jobs in J' and a feasible RNI-schedule finishing at time $\ell_{\max}$ for the jobs in J''. Then we put the remaining singles by using first-fit. For an illustration, see Fig. 2. Clearly, if we succeeded at all steps, the resulting schedule is normal. □

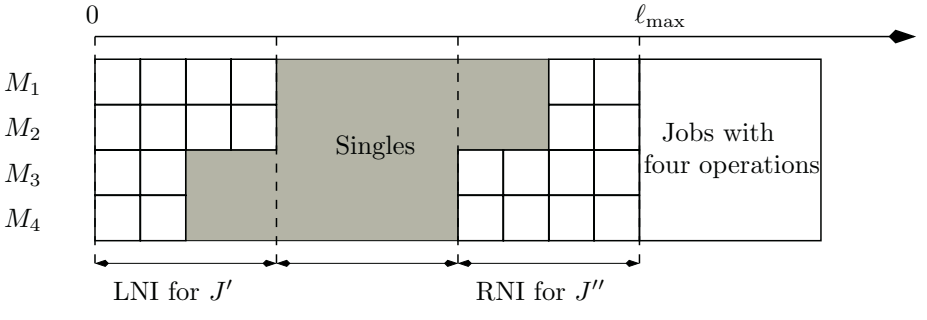


Fig. 2. Scheme of the normal schedule S for 4 machines

It is clear that given an instance of the OB-problem, we may consider it as an instance of the Open Shop problem and provide for it a feasible **open shop schedule** in which no two operations of the same job overlap. Such a schedule is also feasible for the original OB-problem, and when it is normal, it is evidently optimal for the OB-problem.

In [10, 9], a few sufficient conditions were derived which, given an instance of the Open Shop problem satisfying one of these conditions, guarantee that a normal schedule for that instance exists and can be found in polynomial time.

These results can be successfully applied to the Open Block problem, enabling one to construct optimal schedules for most instances in polynomial time.

Theorem 2. *Any instance of the* $OB||C_{\max}$ *problem with* n *jobs and* m *machines satisfying at least one of the inequalities*

$$\ell_{\max} \geq \left(\frac{16}{3}\, m \log_2 m + \frac{61}{9}\, m - 7.4\right) p_{\max}, \tag{3}$$

$$\ell_{\max} \geq m^2 p_{\max}, \tag{4}$$

has a normal schedule exists which can be found in $O(\min\{nm + m^4 \log^2 m, nm^2 \log m\})$ *time in the case of (3), and in* $O(\min\{nm + m^8, n^2 m^2\})$ *time in the case of (4).* □

Remark 1. One can observe from the above theorem that if all job processing times and the number of machines are bounded from above while the number of jobs infinitely increases and the mean job processing time is positive, then $\ell_{\max}$ tends to infinity and the conditions of Theorem 2 are satisfied with high probability. This implies that most instances of the OB-problem are normal and optimal schedules for such instances can be found by one of the polynomial time algorithms of Theorem 2.

Another interesting polynomially solvable case of the OB-problem is formulated in Theorem 3. It is used in Section 5 in designing an approximation algorithm.

Theorem 3. *Every instance of the OB-problem in which the processing times of all jobs are integral degrees of* 2 *is normal. Its optimal (normal) schedule can be found for the problem with* n *jobs and* m *machines in* $O(mn + n \log n)$ *time.* □

The following LNIS-algorithm constructs the desired normal schedule.

LNI Scheduling (LNIS):

Step 1. Number the jobs in nonincreasing order of their processing times: $p_1 \geq p_2 \geq \cdots \geq p_n$.
Step 2. Considering the jobs in this order, schedule each operation of the current job as early as possible, without any idle time on the corresponding machine.

One can describe a similar RNI Scheduling algorithm (RNIS) that constructs at Step 2 an RNI-schedule finishing at point $\ell_{\max}$ (instead of the LNI-schedule being produced by the LNIS-algorithm).

4 Hardness and Inapproximability Results

First we formulate basic NP-complete problems which will be used in our NP-hardness proofs.

PartitionM
Instance: n positive integers $e_1, \ldots, e_n \in \mathbb{Z}^+$ and and an integer $E \in \mathbb{Z}^+$ such that $\sum_{i=1}^{n} e_i = 9E$.
Question: Is there a subset of indices $N \subset \{1, \ldots, n\}$ such that

$$\sum_{i \in N} e_i \;=\; 6E\,? \tag{5}$$

Monotone-Not-All-Equal-Different-4Sat (Modiff-4Sat)
Instance: A set U of variables, a collection C of clauses over U such that each clause has size exactly 4 and contains only unnegated different variables.
Question: Does there exist a truth assignment for U such that each clause in C has at least one true variable and at least one false variable?

Theorem 4. *The $OB6||C_{\max}$ problem is NP-hard.*

Proof. We prove that it is to NP-complete to verify whether a given instance of the 6-machine OB-problem has a normal schedule. To this end we construct a reduction from the PartitionM problem to our OB-problem. Given an instance $\{e_1, \ldots, e_n;\ E\}$ of the PartitionM problem satisfying $\sum_i e_i = 9E$, we define an instance I^* of the OB-problem as follows.

We introduce 6 machines $M_1, \ldots, M_6$ and 8 *basic* jobs: *big jobs* b_1, b_2, b_3, *medium jobs* m_1, m_2, and *small jobs* s_1, s_2, s_3 with processing times:

$$p\,(b_1) = 70,\ p\,(b_2) = 73,\ p\,(b_3) = 67, \quad p\,(m_1) = 17,\ p\,(m_2) = 20,$$

$$p\,(s_1) = 13,\ p\,(s_2) = 10,\ p\,(s_3) = 7.$$

The machines are prescribed to execute the following jobs:

$$M_1:\ \{b_1, m_1, s_1\}, \quad M_3:\ \{b_2, m_1, s_2\}, \quad M_5:\ \{b_3, m_2, s_1\},$$

$$M_2:\ \{b_1, m_2, s_2\}, \quad M_4:\ \{b_2, m_2, s_3\}, \quad M_6:\ \{b_3, m_1, s_3\}.$$

Note that the load ℓ_i of each machine M_i $(i = 1, \ldots, 5)$ is equal to 100, while the current load of M_6 is 91. In addition to operations of basic jobs, machine M_6 is prescribed to execute n *tiny* singles $\{t_1, \ldots, t_n\}$. Each job t_i has length e_i/E. Thus, the total load of machine M_6 is also 100, and so we have $\ell_{\max} = 100$.

To prove the theorem, it suffices to show that the question in the PartitionM problem has the positive answer, if and only if the instance I^* has a normal schedule.

$\Leftarrow$ Assume that there exists a normal schedule S. Since $p\,(b_j) > \ell_{\max}/2$, the operations of each big job b_j must be executed as a block. Since no two different jobs have equal processing times, b_1 and b_2 cannot be executed in the middle of the makespan $[0, \ell_{\max}]$. So the blocks of both jobs can only be scheduled either at the beginning, or at the end of the makespan. W.l.o.g., we may assume that b_1 is processed first on M_1 and M_2.

We first prove that b_2 must be the last job on M_3 and M_4. Assume the contrary. Then b_2 must be the first job on M_3 and M_4. Since $p(m_1) > \{p(s_1), p(s_2)\}$ and $p(b_1) \neq p(b_2)$, job m_1 has to be the last on both M_1 and M_3. Thus, s_2 is in the middle on M_3, and so it must be the last on M_2, while m_2 occurs there in the middle. But in this case we cannot place job m_2 on M_4 properly. This proves our claim.

We now consider three possible locations of job b_3 on machine M_6.

Case 1: b_3 is the third on M_6 (see Fig. 3). Then it is also the third on M_5. Since $2p(m_2) + p(b_3) > \ell_{\max}$ and $p(s_1) \neq p(s_3)$, job m_2 can be the second neither on M_4, nor on M_5. Therefore, it must be the first on both machines whereas s_3 on M_4 must be the second.

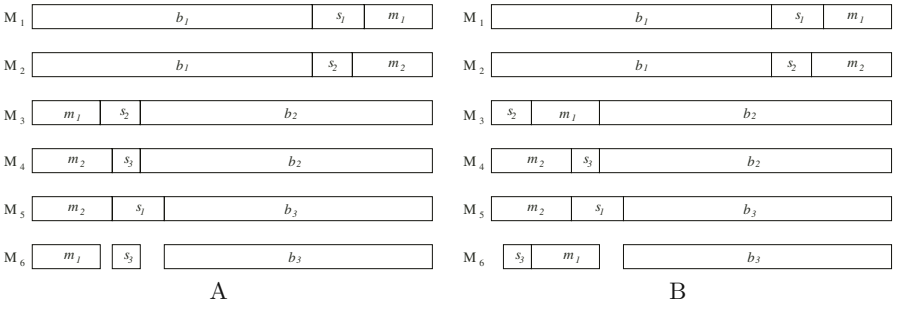


Fig. 3. Normal schedules for the basic jobs in Case 1

If m_1 is the first on M_3 (Fig. 3A), then due to the inequality $2p(m_1)+p(b_3) > \ell_{\max}$, it also must be the first on M_6. Therefore, s_3 must be scheduled on M_6 exactly in the interval $[20, 27]$, so as to form a block with its operation on M_4. This implies the existence of a subset N satisfying (5).

If m_1 is the second on M_3 (Fig. 3B), it must constitute a block (scheduled in the interval $[10, 27]$) with its operation on M_6. The tiny jobs scheduled on M_6 between jobs m_1 and b_3 evidently determine the desired subset N satisfying (5).

Case 2: b_3 is the second on M_6 (see Fig. 4). Then it is also the second on M_5. If m_2 is the first on M_5, then, due to $p(m_1) > p(s_1)$, job m_1 must be the first on M_6 (see Figure 4A). Symmetrically, if m_2 is the third on M_5, then m_1 is the third on M_6 as well (see Figure 4B). In both cases the existence of a subset N satisfying (5) is evident.

Case 3: b_3 is the first on M_6 (see Figure 5). Then it is also the first on M_5.

Clearly, m_2 must be the third on M_2 and M_5, and therefore, s_1 is the second on M_5. This implies that s_1 can only be the third on M_1 whereas m_1 is the second. This, in turn, implies that m_1 is the second job on M_6 constituting a block with its operation on M_1. This means that it must be scheduled in the interval $[70, 87]$, leaving for job s_3 the only opportunity to be scheduled

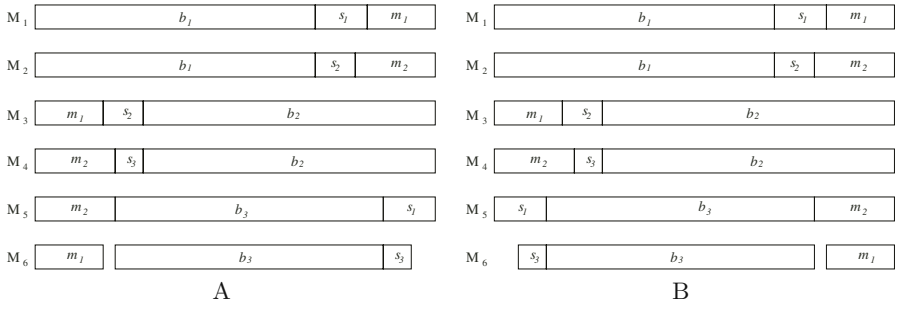


Fig. 4. Normal schedules for the basic jobs in Case 2

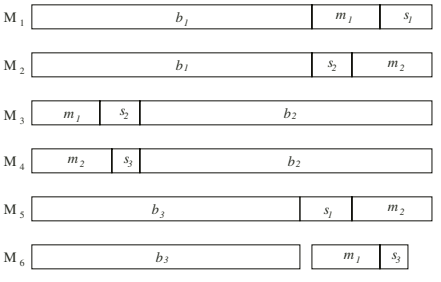


Fig. 5. A normal schedule for the basic jobs in Case 3

somewhere in the interval $[87, 100]$. Thus, the total length of tiny jobs processed on M_6 after m_1 is equal to 6, which implies the answer "yes" to the question of the PARTITIONM problem.

$\Rightarrow$ Suppose that the answer to the question in the PARTITIONM problem is positive. Then a normal schedule for the instance I^* can easily be produced. Indeed, any of the 5 schedules depicted in Figures 3–5 can be used for this purpose. □

A proof of the following problem will appear in a journal version.

Theorem 5. *The* $OB2|r_j|C_{\max}$ *problem is NP-hard in strong sense.* □

Remark 2. By a similar argument it can be shown that it is strongly NP-complete to establish the existence of a feasible schedule for a given instance of the two machine OB-problem with deadlines.

Now we turn to the following decision problem: given an instance of the OB-problem, does there exist a schedule of length at most 10?

Theorem 6. *The problem of deciding if there is a feasible OB-schedule of length at most* 10 *is NP-complete.*

Proof (sketch). To prove that the decision problem is NP-complete, we construct a reduction from the MODIFF-4SAT problem. The technique used in our

reduction differs from the one used in Williamson et. al. [15] in the following: instead of associating jobs and machines with literals (which obliges one to synchronize the jobs corresponding to literals of the same variable), we associate them with variables. □

As a straightforward corollary of the above theorem, we obtain

Theorem 7. *For the OB-problem with a variable number of machines and for any $\rho < 11/10$, there is no polynomial time ρ-approximation algorithm, unless $\mathbb{P} = \mathbb{NP}$.* □

Corollary 1. *For the OB-problem with a variable number of machines, there is no PTAS, unless $\mathbb{P} = \mathbb{NP}$.* □

Now we should explain why in Theorem 6 we consider the time interval of length 10. (Clearly, if we could prove a similar result for a smaller value of $\ell_{\max}$, we would obtain a better lower bound on non-approximability.) The answer is contained in

Theorem 8. *There is a polynomial time algorithm that given an instance of the OB-problem with $\ell_{\max} \leq 9$, constructs a normal (and therefore optimal) schedule.*

Proof. We show this for $\ell_{\max} = 9$. Other cases are easy and can be proved in a similar way. In fact, it suffices for each $i \in \{2, \ldots, 9\}$ to prescribe a set of *allowed locations* for i-jobs within the interval $[0, 9]$, so that:
(a) no two locations of i-jobs overlap in time;
(b) for any feasible combination of jobs destined to be processed on one machine (i.e., such that their total length is at most 9), it should be shown that there exists a schedule for that machine in which each i-job is assigned to one of its allowed locations. Define a set L_i of allowed locations for each $i \in \{2, \ldots, 9\}$ as follows: $L_9 = \{[0, 9]\}$, $L_8 = \{[1, 9]\}$, $L_7 = \{[2, 9]\}$, $L_6 = \{[3, 9]\}$, $L_5 = \{[4, 9]\}$, $L_4 = \{[0, 4], [4, 8]\}$, $L_3 = \{[0, 3], [3, 6], [6, 9]\}$, $L_2 = \{[0, 2], [2, 4], [4, 6], [6, 8]\}$ (clearly, 1-jobs may be scheduled arbitrarily). It can be easily checked that, given a family of jobs $\{J_1, \ldots, J_k\}$ with total length ≤ 9 containing a "long job" (of length ≥ 5), it can be scheduled without overlapping within the time interval $[0, 9]$, so that each job gets one of its allowed locations. The remaining combinations of jobs with total length 9 can be sequenced as follows: $4+4+1$, $4+2+3$, $4+2+2+1$, $3+3+3$, $2+1+3+3$, $2+2+2+3$, $2+2+2+2+1$. □

5 Approximation Algorithms

In this section we present a $\sqrt{2}$-approximation algorithm. We present also a linear time PTAS for the case of an arbitrary fixed number of machines.

First, we describe the *First gluing procedure*. The *First gluing procedure* combines all jobs in each v-class into a single job, with v as the machine set

and $p(v) = \sum_{J \in \boldsymbol{J}(v)} p_j$ as the processing time of the new job. Using a *balanced binary tree* T with at most 2^m nodes (corresponding to classes $\boldsymbol{J}(v)$, $v \subseteq \{1, \ldots, m\}$) and with the length of each path from the root to a leaf no greater than $O(m)$ we can prove that the First gluing procedure can be implemented in $O(m^2 n)$ time and completes with at most $n' = \min\{n, 2^m\}$ aggregated jobs having pairwise different machine sets.

Squared-Round-and-Divide Scheduling (SRDS):

Step 1. Applying the *First gluing procedure*, we get an instance with $n' \leq \min\{n, 2^m\}$ aggregated jobs $\{\hat{J}_j\}$ with processing times $\hat{p}_j$. As we noted, this can be done in $O(m^2 n)$ time.
Step 2. Round the squared processing time $\hat{p}_j^2$ of each aggregated job $\hat{J}_j$ up to the nearest value 2^{α} for an integer α_j. Divide the whole set $\hat{\boldsymbol{J}}$ of aggregated jobs into two subsets: $\boldsymbol{J}_{\text{odd}}$ (the set of jobs with odd exponents α_j) and $\boldsymbol{J}_{\text{even}}$ (with even exponents α_j). Define instance I'_{odd} by jobs $\{J_j \in \boldsymbol{J}_{\text{odd}}\}$ with new processing times $p_j' = 2^{(\alpha - 1)/2}$ and instance I''_{even} by jobs $\{J_j \in \boldsymbol{J}_{\text{even}}\}$ with new processing times $p_j'' = 2^{\alpha/2}$. Compute the loads ℓ_i' and ℓ_i'' of each machine M_i in the instances I'_{odd} and I''_{even}.
Step 3. Applying the LNIS-algorithm, construct the LNI-schedule S_1' for the instance I'_{odd}.
Step 4. Retaining the job order on each machine and the block order for each job as in schedule S_1', find the earliest possible schedule S_1 for the jobs $\boldsymbol{J}_{\text{odd}}$ with the original processing times. Compute the completion time C_i of each machine M_i in this schedule.
Step 5. Compute the parameter $T = \max_{M \in \boldsymbol{M}}(C_i + \ell_i'')$.
Step 6. Applying the RNIS-algorithm, construct the RNI-schedule $S_2(T)$ (finishing at time T) for the instance I''_{even}. Take the same starting times for the jobs $J_j \in \boldsymbol{J}_{\text{even}}$ with the original processing times.

Theorem 9. *For any instance of the OB-problem, there exists a feasible schedule S with length $C_{\max}(S) < \sqrt{2}\,\ell_{\max}$; the schedule can be found by the SRDS-algorithm in $O(m^2 n)$ time, where m is the number of machines and n is the number of jobs.*

Proof. It is clear that all processing times $\{p_j'\}$ and $\{p_j''\}$ in the instances I'_{odd} and I''_{even} are integer degrees of 2, and hence (by Theorem 3), the LNIS and RNIS algorithms produce feasible no-idle schedules S_1' and $S_2(T)$ for the instances I'_{odd} and I''_{even} respectively. When we return to the original processing times, we obtain a schedule S_1 for the jobs in $\boldsymbol{J}_{\text{odd}}$; it is feasible, because we keep the blocks of each job and their mutual order unchanged. For the jobs in $\boldsymbol{J}_{\text{even}}$ we retain the same schedule $S_2(T)$ (i.e., the same starting times of all operations). Therefore, it also remains feasible and keeps the property that on each machine M_i jobs in $\boldsymbol{J}_{\text{even}}$ do not overlap with jobs in $\boldsymbol{J}_{\text{odd}}$. Thus, the resulting schedule for the jobs with original processing times is completely feasible. To complete the proof of the theorem, it remains to show that $T \leq \sqrt{2}\,\ell_i$ for each machine M_i.

Since $T = C_i + \ell_i''$ for some $M_i \in \boldsymbol{M}$, it suffices to show that for each machine M_i we have $C_i \le \sqrt{2}\bar{\ell}_i'$ and $\ell_i'' \le \sqrt{2}\bar{\ell}_i''$, where $\bar{\ell}_i'$ and $\bar{\ell}_i''$ stand for the loads of machine M_i on the sets of jobs $\boldsymbol{J}_{\rm odd}$ and $\boldsymbol{J}_{\rm even}$ in terms of their original processing times. While the second inequality is a simple consequence of our rounding technique, the first one needs a less trivial argument.

The desired inequality follows from two properties: that $\ell_i' \le \bar{\ell}_i'$ (which is clear), and that the completion time of each operation in schedule S_1 exceeds that in schedule S_1' by a factor less than $\sqrt{2}$, which can be proved by the induction on the job number and the block number (assuming that the jobs are numbered in the order of their consideration by the LNIS-algorithm, *i.e.*, in non-increasing order of processing times p_j', and the blocks of each job are numbered according to their starting times in schedule S_1').

Indeed, suppose that the above property holds for jobs $J_1, \ldots, J_{k-1}$ and for earlier blocks of job J_k, and we are scheduling the next block (B) of job J_k. Let the block consist of operations $O_1, \ldots, O_s$ that are to be scheduled on machines $M_1, \ldots, M_s$ respectively (for simplicity of notation). Since the starting time of block B is chosen to be as small as possible, it coincides with the maximum of the completion times of machines $M_1, \ldots, M_s$ on the set of jobs $\{J_1, \ldots, J_{k-1}\}$ and the completion time of the previous block of job J_k (if any). In any case, it coincides with the completion time of some previously scheduled operation, and by the induction hypothesis, is at most $\sqrt{2}$ times the completion time of schedule S_1'. Since the length of block B is also less than a factor of $\sqrt{2}$ of the length of that block for the rounded instance I', the required property follows.

To complete the proof of the theorem, it remains to estimate the running time of the algorithm. It is clear that (not counting step 1) most costly steps of the algorithm are steps 4 to 6, where we have to apply the ordering procedure. Since the latter requires time $O(n' \log n')$ and $n' \le \min\{n, 2^m\}$, this reduces to $O(nm)$ time, and so, the overall running time of the algorithm coincides with that of step 1 (namely, $O(nm^2)$). □

By Theorem 6 no PTAS can be designed for the OB-problem (unless $\mathbb{P} = \mathbb{NP}$) in the case when the number of machines is treated as a variable. An opposite result can be derived, if we treat the number of machines as a constant. In this case, a linear time PTAS exists similar to that proposed by Sevastianov and Woeginger [13] for the Open Shop problem.

PTAS for the OB-Problem with a Fixed Number of Machines. For any fixed $\varepsilon > 0$, we present an algorithm A_ε that outputs a schedule S with makespan $C_{\max}(S) \le (1+\varepsilon)\,OPT$. The desired $(1+\varepsilon)$-approximation can be easily attained in the case that $d_{\max} \le \varepsilon \cdot \ell_{\max}$: a simple greedy algorithm [13] finds a schedule S with makespan

$$C_{\max}(S) \le \ell_{\max} + d_{\max} \le (1+\varepsilon)\ell_{\max} \le (1+\varepsilon)OPT$$

in time $O(m^2 n)$. The running time can be reduced to $O(nm + C(m,\varepsilon))$, if we first apply the following *Second gluing procedure* with $\boldsymbol{d} = \varepsilon \cdot \ell_{\max}$. Given a *lower threshold* $\boldsymbol{d}$ on lengths of jobs, we scan the list of jobs, gluing two jobs

together each time that the maximum of their lengths is no greater than $\boldsymbol{d}/2$. The procedure terminates, as soon as each job in the resulting instance (but maybe one job) has length greater than $\boldsymbol{d}/2$.

In the case $d_{\max} > \varepsilon \cdot \ell_{\max}$ we divide the set of jobs $\boldsymbol{J}$ into three subsets L, M, S of *large, medium*, and *small* jobs, respectively: $L = \{J_j \in \boldsymbol{J} \,|\, d_j \geq \alpha' \ell_{\max}\}$, $M = \{J_j \in \boldsymbol{J} \,|\, \alpha'' \ell_{\max} \leq d_j < \alpha' \ell_{\max}\}$, $S = \{J_j \in \boldsymbol{J} \,|\, d_j < \alpha'' \ell_{\max}\}$. The values of α' and α'' are chosen with respect to the value of ε so that:

(a) the number $|L|$ of large jobs was bounded above by a constant;
(b) the total length of medium jobs was at most $\varepsilon \cdot \ell_{\max}$;
(c) the ratio α''/α' was small enough, so as to meet $\alpha' + \alpha'' + \alpha''|L| \leq \varepsilon$.

As proved in [13], the numbers α' and α'' with the above properties exist, the value of α'' being at least $\alpha^* \doteq \varepsilon/(e^{1.25} \cdot 2^{m/\varepsilon})$. The latter bound enables us to reduce the total number of jobs to at most a constant number $2m/\alpha^*$ by applying the *Second gluing procedure* with the threshold $\boldsymbol{d} = \alpha^* \ell_{\max}$. Thus, we come to the following algorithm A_ε in the case $d_{\max} > \varepsilon \cdot \ell_{\max}$.

ALGORITHM A_ε (in the case $d_{\max} > \varepsilon \cdot \ell_{\max}$):

Step 1. Apply the *Second gluing procedure* with $\boldsymbol{d} = \varepsilon \cdot e^{-1.25} \cdot 2^{-m/\varepsilon} \cdot \ell_{\max}$. Number the jobs with $d_j > \boldsymbol{d}$ in nonincreasing order of d_j.
Step 2. Find a partition of the job set into subsets L, M, and S satisfying (a)–(c).
Step 3. Construct an optimal schedule $OPT(L)$ for the jobs in L.
Step 4. Use the greedy algorithm [13] to place the jobs of M and S into $OPT(L)$.

Similar to [13], the following result can be proved.

Theorem 10. *For any $\varepsilon > 0$ and any instance of the $OB||C_{\max}$ problem with n jobs and m machines, algorithm A_ε constructs a schedule S with makespan*

$$C_{\max}(S) \leq (1 + \varepsilon) OPT.$$

The algorithm runs in time $O(nm + C(m, \varepsilon))$, where $C(m, \varepsilon)$ is the time needed to find an optimal schedule for at most $O(2^{m/\varepsilon})$ so-called "large" jobs.

Acknowledgement

The authors would like to express their much gratitude to Olga Berger for her kind assistance in preparing the paper.

References

1. AMOURA, A.K., BAMPIS, E., KENYON, C., AND MANOUSSAKIS, Y. Scheduling independent multiprocessor tasks. In *Proceedings 5th European Symposium on Algorithms* (1997), LNCS 1284, Springer Verlag, pp. 1–12.

2. Bampis, E., Caramia, M., Fiala, J., Fishkin, A. V., and Iovanella, A. On scheduling of independent dedicated multiprocessor tasks. In *Proceedings 13th International Symposium on Algorithms and Computation* (Vancouver, BC, Canada, 2002), LNCS 2518, Springer Verlag, pp. 391–402.
3. Fiala, T. An algorithm for the open-shop problem. *Math. Oper. Res. 8* (1983), 100–109.
4. Fishkin, A.V., Jansen, K., and Porkolab, L. On minimizing average weighted completion time of multiprocessor tasks with release dates. In *Proceedings 28th International Colloquium on Automata, Languages and Programming* (Crete, 2001), LNCS 2076, Springer Verlag, pp. 875–886.
5. Garey, M.R., and Johnson, D.S., Computers and intractability: A guide to the theory of NP-completeness, *Freeman, San Francisco, CA*, 1979.
6. Gonzalez, T., and Sahni, S. Open shop scheduling to minimize finish time. *Journal of the ACM 23* (1976), 665–679.
7. Hoogeveen, J.A., de Velde, S.L.V., and Veltman, B. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Appl. Math. 55* (1994), 259–272.
8. Rouskas, G.N. *Optical WDM networks: Principles and practice.* Kluwer Academic Publishers, 2000, ch. Scheduling algorithms for unicast, multicast, and broadcast.
9. Sevast'janov, S.V. Polynomially solvable case of the open-shop problem with arbitrary number of machines. *Cybernet. Systems Anal. 28* (1992), 918–933. (Translated from Russian.)
10. Sevast'janov, S.V. Vector summation in Banach space and polynomial algorithms for flow shops and open shops. *Math. Oper. Res. 20* (1995), 90–103.
11. Sevastianov, S. Nonstrict vector summation in multi-operation scheduling, *Annals of Oper. Res. 83* (1998), 179–211.
12. Sevastianov, S.V., and Woeginger, G.J. Makespan minimization in open shops: A polynomial time approximation scheme. *Mathematical Programming 82* (1998), 191–198.
13. Sevastianov, S.V., and Woeginger, G.J. Linear time approximation scheme for the multiprocessor open shop problem. *Discrete Appl. Math. 114* (2001), 273–288.
14. Thaker, D., and Rouskas, G.N. Multi-destination communication in broadcast WDM networks: A survey. Tech. Rep. 2000-08, North Caroline State University, 2000.
15. Williamson, D.P., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevastianov, S.V., and Shmoys, D.B. Short shop schedules. *Oper. Res. 45* (1997), 288–294.

Randomized Priority Algorithms
(Extended Abstract)

Spyros Angelopoulos

Department of Computer Science, University of Toronto
Toronto, Ontario, Canada M5S 3G4
spyros@cs.toronto.edu

Abstract. In a paper of Borodin, Nielsen and Rackoff [8], a framework for abstracting the properties of deterministic greedy-like algorithms was proposed. We extend their model so as to formally define "randomized greedy-like algorithms" and be able to prove lower bounds on the approximability of a certain problem by such a class of algorithms. We show how our techniques can be applied in well-studied problems such as the facility location and makespan scheduling problems, for which both upper and lower bounds on the approximation ratio achieved by deterministic greedy-like algorithms are known.

1 Introduction

Borodin Nielsen and Rackoff [8] presented a framework for abstracting the main properties shared by several "greedy and greedy-like" deterministic algorithms. Within the proposed framework, they showed how to derive lower bounds on the approximability of various scheduling problems by such a class of algorithms. In this paper we extend the "priority algorithm" paradigm in [8] so as to define *randomized* greedy and greedy-like algorithms. More precisely, we show how to use the *Von Neumann/Yao principle* in the context of priority algorithms to prove lower bounds for algorithms that behave in a greedy-like fashion but also allow randomization. We apply the technique to two well-studied NP-hard problems: the problem of makespan scheduling, and the problem of metric facility location.

According to [8], deterministic greedy-like algorithms are classified as (deterministic) priority algorithms, namely algorithms with the following two properties: i) The algorithm selects an ordering of the "input items", and considers each input item in the determined order; and ii) As each input item is considered, the algorithm makes an "irrevocable decision" concerning the input item. Here, the decision on how to represent an input (i.e., what the input items are) and the precise meaning of an irrevocable decision are pertinent to specific problems.

Depending on whether the ordering of input items can change throughout the execution of the algorithm, we define two distinct classes of priority algorithms. Algorithms in the class Fixed Priority commit to an ordering prior to considering any input item, and the ordering cannot change in any subsequent iteration. On the other hand, algorithms in the more general Adaptive

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 27–40, 2004.
© Springer-Verlag Berlin Heidelberg 2004

PRIORITY class are allowed to specify a new ordering after each input item is considered and processed. Following the definition of [8], a (fixed or adaptive) priority algorithm belongs in the class GREEDY if the irrevocable decision obeys the rule that the objective function is *locally optimized*; specifically, the objective function must be optimized as if the currently considered input item were the last one.

There are two natural ways to introduce randomization in a priority algorithm. First, the algorithm may choose a *random ordering* of the input items. Second, the algorithm can make *random decisions* for each input item[1]. In the case where randomization can be applied in both orderings and decisions we refer to such algorithms as *fully randomized algorithms*.

In the online world it is a well-known fact that randomization can be helpful in improving the competitiveness of algorithms. As an illustrative example, we mention the randomized algorithm of Bartal *et al.* [4] for makespan scheduling on two machines which achieves the best possible competitive ratio, namely $\frac{4}{3}$ (in contrast there is a $\frac{3}{2}$ lower bound on the competitive ratio of every deterministic online algorithm). Likewise, there is evidence that randomization may be useful in improving the approximation ratio of priority algorithms. For instance, Borodin *et al.* [8] presented a "classify and randomly select" *priority* algorithm for the problem of interval scheduling, and showed that it is a $O(\log \Delta)$ approximation (here Δ is the ratio of the maximum profit per unit size over the minimum profit per unit size amongst all jobs in the input). In contrast, they showed that, for interval scheduling in a single machine, no deterministic priority algorithm can achieve better than a Δ approximation.

Our main goal is to show that it is possible to derive interesting lower bounds on the approximability of optimization problems by randomized priority algorithms[2]. While the technique is general enough to address full randomization, it can also be applied to algorithms with more restricted access to randomness, i.e., randomized orderings or randomized decisions only. We apply the framework to the following well-known problems:

1. The **metric facility location problem**. **(a)** For the model in which the input is represented as a set of facilities, we show that no fully-randomized priority algorithm is better than a $\frac{4}{3}$ approximation. For randomized-ordering FIXED PRIORITY GREEDY algorithms, we prove a 1.5 lower bound on the approximation ratio. **(b)** For the model in which the input is described as a set of cities, we show that no fully-randomized priority algorithm is better than a 2-approximation.

2. The **makespan scheduling problem** on identical machines. We show that no fully-randomized priority algorithm achieves an approximation ratio better than $\frac{12}{11}$. For FIXED PRIORITY algorithms with randomized decisions, and when there are at last three machines, we show a corresponding lower bound of $\frac{10}{9}$.

[1] In particular, every randomized online algorithm can be classified as a FIXED PRIORITY, random-decisions algorithm.

[2] Due to space limitations we omit proofs of certain theorems and lemmas in this extended abstract. Full proofs will appear in the journal version of the paper.

We emphasize that, as in deterministic priority algorithms (and similar to competitive analysis in the online world) the lower bounds we derive are orthogonal to any complexity considerations. In other words, we allow the algorithm unbounded time complexity.

As one may expect, the criterion of what precisely constitutes a greedy-like algorithm can be very subjective. Hence one can argue that there exist algorithms which could intuitively be characterized as (deterministic or randomized) greedy algorithms, and which do not fit in the priority algorithm framework, e.g., greedy-like algorithms that apply iterative refinements of partial solutions, or greedy-like algorithms which can make temporary decisions about each input item, i.e., decisions that can be revoked in future iterations (an example of an algorithm in the latter class is the one-pass *Admission* algorithm of Bar-Noy, Guha, Naor and Schieber [3]). However, we claim that the priority-algorithm model still captures a rich class of (deterministic or randomized) algorithms that can be classified as greedy-like. To support the claim, Section 3 overviews some upper bounds for the problems we consider, and which are achieved by priority algorithms.

Clearly, this line of work is motivated by the fact that greedy algorithms are a standard tool in both theory and practice, and thus it is useful to know in which cases the greedy approach will not perform efficiently. There are two more reasons we believe it is interesting to focus on the approximation power of the broad class of priority algorithms (as opposed to only considering a few "natural" greedy algorithms for every specific problem). First, it is possible that the intuition behind the lower-bound constructions could provide insight into how to design better randomized priority algorithms. Similar insight has proven quite useful in the design of online algorithms. For a concrete example, see [28] where the proof that no randomized online algorithm for 2-machine scheduling is better than $\frac{4}{3}$-competitive suggests how to design a randomized algorithm which is $\frac{4}{3}$-competitive (a result due to Bartal *et al.* [4]).

Second, it is not always the case that the best approximation is achieved by natural greedy-like (deterministic or randomized) algorithms. Leaving aside the debatable issue of characterizing a natural greedy algorithm (arguably, a very subjective task), there exist problems where good approximations are achieved by greedy-like algorithms which, if not "unnatural", certainly are not straightforward. For instance, the natural greedy algorithm of Mahdian *et al.* [21] for metric facility location does not achieve as good an approximation ratio as the priority algorithm of Jain, Mahdian and Saberi [20], which is described as a primal-dual algorithm and does not yet have a clear combinatorial statement. A similar situation has been observed in online computation: for instance, some of the recent algorithms for makespan scheduling (see, e.g., [1]), as well as algorithms for load balancing in related and unrelated machines (see [6] for a detailed exposition) are definitely not among the most intuitive approaches one could come up with. Since every online algorithm is also a Fixed Priority algorithm, it is reasonable to believe that not every greedy-like algorithm is expected to have a natural statement.

2 Preliminaries

Deterministic vs. Randomized Priority Algorithms and the Minimax Principle. Informally, we can define a randomized priority algorithm as a probability distribution over the set of all deterministic priority algorithms *of the same class*. To make the definition more precise, we first provide a more formal description of deterministic priority algorithms. Consider a specific optimization problem Π; one can associate with Π a set $\mathcal{S}$ of **input items**. For instance, for the problem of makespan scheduling, $\mathcal{S}$ is the (infinite) set of all possible jobs, each characterized by its size and its id. For every input item $s \in \mathcal{S}$, denote by $D(s)$ the set of **allowable decisions** associated with s; e.g., in makespan scheduling with m machines, $D(s)$ can be described by the set $\{1, \ldots, m\}$, where decision i can be interpreted as "schedule input item s on machine i".

A deterministic priority algorithm for Π is specified by two functions: the **ordering function** and the **decision function**. The ordering function is simply a function that produces an ordering of the input items in S. As observed in [8], one has to restrict the ability of the algorithm to produce an ordering, otherwise the algorithm can choose an ordering that can give rise to an optimal solution. The nature of the adversary described in [8] provides an (implicit) restriction on the types of ordering functions that are allowed.

On the other hand, the decision function is a function which, given the current input item $s \in \mathcal{S}$, and the **history** of the algorithm's execution, maps s to a decision in $D(s)$. Here, the term history refers to all inputs the algorithm has considered in the past (and implicitly, to the algorithm's corresponding decisions).

A priority algorithm $\mathcal{A}$ for a cost-minimization problem is a *c-approximation* algorithm[3] if

$$\mathcal{A}(\sigma) \leq c \cdot OPT(\sigma), \tag{1}$$

for every input σ, where $\mathcal{A}(\sigma)$, $OPT(\sigma)$ denote the cost of algorithm $\mathcal{A}$ and the optimal cost on input σ, respectively. The approximation ratio of $\mathcal{A}$ is defined as the infimum of the set of values c for which $\mathcal{A}$ is a c-approximation.

A randomized priority algorithm is a priority algorithm in which either the ordering function or the decision function, or both functions are randomized; we refer to such algorithms as randomized on the orderings, randomized on the decisions, or fully-randomized, respectively. For a randomized priority algorithm, the definition of a c-approximation is as in (1), with $\mathcal{A}(\sigma)$ replaced by its expected value over the random choices of $\mathcal{A}$, denoted by $\mathbf{E}[\mathcal{A}(\sigma)]$.

As shown by Borodin *et al.* [8], one can derive lower bounds for **deterministic** priority algorithms by evaluating the performance of every such algorithm

[3] Alternatively, $\mathcal{A}$ is an *asymptotic* c-approximation algorithm if $\mathcal{A}(\sigma) \leq c \cdot OPT(\sigma) + o(OPT(\sigma))$. This is a weaker definition of approximability, since it ignores the (sometimes pathological) performance of the algorithm on small input sets. The Von Neumann/Yao principle is still applicable, however one has to supplement (2) with additional constraints, similar to the argument of Chrobak *et al.* [10] in the context of competitive analysis of online algorithms (details will be given in the full paper).

on an appropriately constructed nemesis input. But how can we show bounds on the approximability of a problem by a **randomized** priority algorithm? First of all, we will assume that the adversary against a randomized priority algorithm has knowledge of the algorithm, but does not have any access to the random choices the algorithm makes in the course of the game. Borrowing the terminology of competitive analysis, we refer to such an adversary as the *oblivious adversary*. Under this assumption, one can resort to the *Von Neumann/Yao principle* to prove lower bounds on the approximation ratio of randomized priority algorithms against such an adversary, similar to the application of the same principle in the analysis of online algorithms (as introduced by Borodin, Linial and Saks [7]). More specifically, suppose that there exists a constant c such that we can present a distribution $\mathcal{D}$ over inputs σ such that

$$\mathbf{E}_{\mathcal{D}}[\mathcal{A}(\sigma)] \geq c \cdot \mathbf{E}_{\mathcal{D}}[OPT(\sigma)], \tag{2}$$

for every deterministic priority algorithm $\mathcal{A}$ in the class $\mathcal{C}$. Then the cost-minimization problem cannot be approximated by any randomized priority algorithm in the class $\mathcal{C}$ within a ratio better than c.

Naturally, the crux in the application of the Von Neumann/Yao principle lies in the selection of the appropriate input distribution. In the case of Adaptive Priority algorithms, as well as fully-randomized Fixed Priority algorithms, one has to resort to ad hoc methods for identifying the desired distribution. A special case arises when one considers Fixed Priority algorithms with randomization on the decisions *only*, in the sense that one can have better insight on how to identify a good distribution. More precisely, the distribution can be determined by a game between a deterministic Fixed Priority algorithm and the adversary, as follows: the adversary presents a set of potential input items S, and having knowledge only of the ordering that is decided by the algorithm, removes input items from S according to a specific probability distribution, so as to generate the distribution over the actual inputs[4].

Problem definitions and input representations. The input to the *makespan scheduling* problem on m identical machines consists of a set of jobs, each having a certain *size*. The *load* of a machine is the sum of the sizes of jobs scheduled on it. The objective is to minimize the maximum load among the m machines.

In the *(uncapacitated, unweighted) facility location* problem, the input consists of a set $\mathcal{F}$ of facilities and a set $\mathcal{C}$ of cities (we restrict our study to the case $\mathcal{F} \cap \mathcal{C} = \emptyset$). Each facility $i \in \mathcal{F}$ is associated with an *opening cost* f_i which reflects the cost that must be paid to utilize the facility. Furthermore, for every facility $i \in \mathcal{F}$ and city $j \in \mathcal{C}$, the non-negative *distance* or *connection cost* c_{ij} is the cost that must be paid to connect city j to facility i. The objective is to

[4] Note that an Adaptive Priority algorithm with randomized decisions will introduce randomization on the selection of the orderings as well: this is due to the fact that the next input to be considered depends on the outcome of processing inputs that were considered in the past. Hence, a similar game is not applicable in Adaptive Priority algorithms.

open a subset of the facilities in $\mathcal{F}$ and connect each city in $\mathcal{C}$ to an open facility so that the total cost incurred, namely the sum of the opening costs and the connection costs, is minimized. We focus on the *metric* version of the problem, in which the connection costs satisfy the triangle inequality.

When considering priority algorithms for the above problems, we need to establish a model for representing the input. For the makespan scheduling problem, there is no issue as to what constitutes the input: the input is a set of jobs, with each job being characterized by its id and its size. In contrast, there exist two natural ways to represent the input to the facility location problem. In particular, we may assume that the input consists of facilities, where each facility is identified by a unique id, its distance to every city and its opening cost. When considering a facility of highest priority, the algorithm must make an irrevocable decision as to whether or not to open this facility. This model of input representation was assumed in [2], since it abstracts several known priority algorithms(see the adaptive-priority algorithms in [21] and [20] as well as a fixed priority algorithm in [23]). We refer to the above representation of inputs as the *facility-input model*.

A different way of representing the input is to view the cities as the input items. Namely, the input consists of the set of all cities, with each city being identified by its id, and its distance to each facility. The opening costs of the facilities are treated as global information. In this context, an irrevocable decision is interpreted as assigning each considered city to some open facility, possibly opening a new facility, but without affecting the assignment of cities considered in the past. We refer to this model of representing the inputs as the *city-input model*.

3 Previous Work

The first constant-factor polynomial-time approximation algorithm for (metric) facility location was given by Shmoys, Tardos and Aardal [30]. The past several years have witnessed a steady series of improvements on the approximability of this problem (the interested reader is referred to the survey of Shmoys [29]). Currently, the best-known approximation ratio (1.52) is due to Mahdian, Ye and Zhang [22], and is achieved by a non-priority algorithm. Guha and Khuller [17] have shown that the problem is not approximable within a factor better than 1.463, unless $NP \subseteq DTIME(n^{O(\log \log n)})$.

We identify algorithms that can be described as priority algorithms. Mahdian, Markakis, Saberi and Vazirani [21] showed that a natural Adaptive Priority algorithm gives a 1.861 approximation; the analysis is performed by an elegant application of the dual-fitting technique. To our knowledge, the best approximation achieved by a priority algorithm (as observed in [2]) is 1.61 and is due to Jain, Mahdian and Saberi [20] (also analyzed by using the dual fitting technique). Mettu and Plaxton [23] showed that an algorithm which belongs in the class Fixed Priority Greedy yields a 3-approximation. Concerning the use of randomization, Meyerson [24] showed that a fully-randomized priority algo-

rithm can achieve a constant-factor approximation. In recent work Fotakis [12] presented a deterministic online algorithm with a competitive ratio of $O(\frac{\log n}{\log \log n})$ (here n denotes the number of nodes, and each node can be both a facility and a city); on the negative side, he showed that no randomized online algorithm can achieve a competitive ratio better than $\Omega(\frac{\log n}{\log \log n})$, against the oblivious adversary.

Makespan scheduling on identical machines is known to be strongly NP-hard [13]. Hochbaum ans Shmoys [18] showed that it is possible to derive a PTAS, while Sahni [25] showed that, in the case where the number of machines m is fixed, there exists a FPTAS for the problem. For the variation of the problem in which jobs arrive on-line, Graham's [15] famous *List Scheduling* algorithm is well-known to be $(2 - \frac{1}{m})$-competitive. Currently, the best competitive ratio achieved by a deterministic algorithm is 1.9201 [11], while the best known lower bound is 1.853 and it is due to Gormley *et al.* [14]. When randomization is introduced, a recent algorithm proposed and analyzed by Albers [1] achieves a competitive ratio of 1.916 for general m. Chen, van Vliet and Woeginger [9] and Sgall [27] showed that no randomized online algorithm can be better than $\frac{1}{1-(1-\frac{1}{e})}$-competitive.

Concerning priority algorithms, Graham [16] showed that a simple algorithm called *Longest Processing Time First* (or LPT, for brevity), which sorts jobs by non-increasing size and assigns the current job to the least loaded machine, is a $\frac{4m-1}{3m}$-approximation; note that LPT belongs in the class Fixed Priority, Greedy. Seiden, Sgall and Woeginger [26] showed that LPT is optimal for $m = 2$ when jobs are considered by non-increasing size (which they call semi-online scheduling); they also presented a $\frac{8}{7}$-approximation semi-online algorithm with random decisions for $m = 2$, and showed it is optimal for the class of semi-online randomized algorithms when $m = 2$. Borodin *et al.* [8] showed that LPT is optimal with respect to all priority algorithms when $m = 2$. They also proved that LPT is optimal in the class Fixed Priority, Greedy, when $m = 4$, and that, for arbitrary m, no (deterministic) priority algorithm is better than a $\frac{7}{6}$ approximation.

4 Randomized Priority Facility Location

In this Section we show lower bounds on the approximability of the metric facility location problem by randomized priority algorithms. All our lower-bound constructions for metric facility location use connection costs in $\{1, 3\}$, suggesting the following definitions. Let C_f be the set of cities at distance 1 from facility f. We say that f *covers* C_f. Two facilities f and f' are called *complementary* if and only if $C_f \cup C_{f'} = C$ (i.e., f and f' cover all cities in the input), $C_f \cap C_{f'} = \emptyset$, and f and f' have the same opening cost. For convenience, we call f' the *complement* of f (and vice versa), and we use the notation $\overline{f}$ to denote the complement of a given facility f.

4.1 Randomized Priority Facility Location in the Facility-Input Model

Theorem 1. *No fully-randomized priority algorithm for facility location has approximation ratio better than* $\frac{4}{3}$.

Proof. Suppose we have $n = 2^k$ cities, each with an id in $\{0, \ldots, 2^k - 1\}$ in binary representation, and k pairs of complementary facilities as the set of potential facilities F. Each facility has an opening cost of $\frac{n}{4}$. The facilities are identified by the cities they cover in the following way: The i-th pair of complementary facilities, denoted by $f_i, \overline{f}_i$ with $i \leq k$ is such that facility f_i covers only cities whose i-th bit is 0, while $\overline{f}_i$ covers only cities whose i-th bit is 1. The distance between a facility and a city it does not cover is equal to 3. By construction, any j facilities in F no two of which are complementary cover exactly $n \sum_{i=1}^{j} 2^{-i} = n(1 - 2^{-j})$ cities.

Let A be a deterministic priority algorithm. We define a distribution over the actual inputs as follows. Only one of the k pairs of complementary facilities in F appears in the actual input, and that pair is chosen uniformly at random, i.e., with probability $\frac{1}{k}$. For each of the remaining $k-1$ pairs, only one of the two facilities will be in the actual input, and it is likewise chosen uniformly at random, i.e., with probability $\frac{1}{2}$. No other facilities are in the actual input. The optimal algorithm will always open the pair of complementary facilities, hence $\mathbf{E}[OPT] = 2\frac{n}{4} + n = \frac{3n}{2}$ (for the remainder of the proof we refer to the two complementary facilities as the *optimal facilities*). If A does not open both optimal facilities, the minimum cost it must pay is $2n$, and is achieved when A opens either two or three facilities (as shown in [2]). Clearly, A will pay at least $2n$ if it opens more than 3 facilities, hence we may assume in what follows that A opens at most 3 facilities.

The intuition behind the proof is that **w.h.p.** A will consider too many facilities for which it cannot have knowledge whether their complement is in the actual input. Since A can open only a small number of facilities, **w.h.p.** it will fail to open both optimal facilities.

Lemma 1. *The expected cost of* A *is at least* $(1 - e^{-\frac{}{144}})(1 - \frac{9}{k}) \cdot 2n$.

Proof. Let H_j be the set of facilities that A has considered before iteration j. Let g_j denote the facility that would receive the highest priority in iteration j, assuming that the remaining set of facilities to be considered was $F \setminus H_j$. For a fixed i, let j_i be the earliest iteration of A in which $g_j = f_i$ or $g_j = \overline{f}_i$ (note that such a j_i always exists). We call the pair of (complementary) facilities $(f_i, \overline{f}_i)$ *bad* if and only if the facility g_j is in the actual input. Note that if the pair $(f_i, \overline{f}_i)$ is bad, then at the iteration in which A considers g_j , it cannot know whether its complement is in the actual input.

Denote by B the set of bad pairs of facilities. Clearly, the pair of optimal facilities is always bad (regardless of the decisions of A), while each of the remaining $k-1$ pairs is bad with probability $\frac{1}{2}$, independently of every other pair.

Hence, we can say that $|B| = 1 + X$, where X is the sum of $k-1$ independent random variables, each of them equal to 1 with probability $\frac{1}{2}$, and 0 with the same probability. We can therefore apply the Chernoff bound, to get that $\Pr(|B| < \frac{k}{3}) < e^{-\frac{k}{144}}$.

Let l denote the number of facilities that A opens, and recall that we can assume $l \leq 3$. Then, the probability A will open both optimal facilities is upper-bounded by the probability of success of any algorithm for the following game: Suppose there are $|B|$ lottery tickets, only one of them being the winning ticket (but the algorithm is oblivious of the winning ticket). The algorithm must go over all tickets, and every time it considers a ticket, either it buys it, or it ignores it; in either case, after the algorithm makes its decision about the current ticket, the adversary discloses the ticket to the algorithm (hence the algorithm gets to know whether the ticket was the winner). Last, the algorithm can buy at most l tickets. Clearly, the probability the algorithm buys the winner is (at most) $\frac{l}{|B|}$. This is also an upper bound on the probability A will open both optimal facilities *conditioned* on the event there exist $|B|$ bad pairs of facilities. Combining the above, we have that with probability at least $(1 - 2e^{-\frac{k}{144}})(1 - \frac{3l}{k})$, A does not open both optimal facilities, hence its total cost is at least $2n$, as argued earlier in the proof. Summarizing, the expected cost of A is lower-bounded by

$$(1 - e^{-\frac{k}{144}})(1 - \frac{3 \cdot 3}{k}) \cdot 2n.$$

□

The theorem follows since for large values of n (and thus for k as well) the ratio $\frac{\mathbf{E}[A]}{\mathbf{E}[OPT]}$ can be made arbitrarily close to $\frac{4}{3}$. □

Theorem 2. *There is a lower bound of* $1.5 - \epsilon$ *on the approximation ratio of every randomized (on the orderings)* Fixed Priority Greedy *facility location algorithm, for arbitrarily small* ϵ.

Proof. Suppose we have a set C of n cities with id's $1 \ldots n$. Define the set F of potential facilities as follows: For a fixed integer $i \in [\frac{n}{2}]$ facility f_i covers cities $1 \ldots \frac{n}{2}$ excluding city i, and also covers city $\frac{n}{2} + i$. The set F consists of all f_i's and $\overline{f}_i$'s, for $i \in [\frac{n}{2}]$, and each facility has an opening cost equal to $2 - \epsilon$, for arbitrarily small ϵ. Recall that for every i, $f_i, \overline{f}_i$ are complementary facilities, and that the distance between a facility and a city not covered by it is equal to 3.

Consider a deterministic Fixed Priority Greedy algorithm A, and denote by I_i (respectively $\overline{I}_i$) the potential input that consists of all facilities of the form $f \in F$ (resp. $\overline{f} \in F$) as well as facility $\overline{f}_i$ (resp. f_i), for $i \in [\frac{n}{2}]$. The actual input to A, which we denote by z, is selected uniformly at random from the set $I \cup \overline{I}$, where $I = \{I_i, i \in [\frac{n}{2}]\}$ and $\overline{I} = \{\overline{I}_i, i \in [\frac{n}{2}]\}$, i.e., a specific actual input is chosen with probability equal to $\frac{1}{n}$. Since every potential input contains (exactly) one pair of complementary facilities, OPT can cover all n cities by opening the two complementary facilities, hence $\mathbf{E}[OPT] = n + 2(2 - \epsilon)$. Denote by A_f the cost of facilities opened by A. Note that

$$\begin{aligned}\mathbf{E}[A_f] &= \Pr(z \in I) \cdot \mathbf{E}[A_f | z \in I] + \Pr(z \in \overline{I}) \cdot \mathbf{E}[A_f | z \in \overline{I}] \\ &= \frac{1}{2}(\mathbf{E}[A_f | z \in I] + \mathbf{E}[A_f | z \in \overline{I}]). \end{aligned} \tag{3}$$

In order to show the wanted, we give expressions for $\mathbf{E}[A_f | z \in I]$ and $\mathbf{E}[A_f | z \in \overline{I}]$; the intuition is that not both expressions can be very small, hence $\mathbf{E}[A_f]$ is large.

Lemma 2. $\mathbf{E}[A_f | z \in I] + \mathbf{E}[A_f | z \in \overline{I}] \geq n(1 - \frac{\epsilon}{2})$.

The theorem follows, since from (3) we have $\mathbf{E}[A_f] \geq \frac{n}{2}(1 - \frac{\epsilon}{2})$, and the expected total cost paid by A is at least $\frac{3n}{2}(1 - \epsilon')$, for arbitrarily small ϵ'. □

4.2 Randomized Priority Facility Location in the City-Input Model

Theorem 3. *No fully-randomized priority algorithm for facility location in the* city-input *model is better than a 2-approximation.*

Proof. The set C of potential input items (cities) consists of n^2 cities. There are $\binom{n^2}{n}$ facilities, each covering (i.e., at distance 1) a set of n cities in C, and no two facilities cover the same set of cities. The distance from a given facility to every city it does not cover is 3, and the opening cost of every facility is equal to 2. Each of the n^2 cities in C will appear in the actual input with probability $\frac{1}{n}$. Thus, the expected number of cities in the actual input is n, and since there is a facility that covers these n cities, $\mathbf{E}[OPT] = n + 5$.

We will bound the expected cost of a deterministic algorithm A. First, note that every time a city c not covered by an opened facility is considered, it is to the algorithm's benefit to open a new facility: the total cost of opening a new facility (say f) that covers c and connecting c to f is 3, which is the connection cost that must be paid if no facility is opened; therefore, we assume that when A considers an uncovered city it must open a new facility (this will only strengthen the bound). To reflect this action, we say that (the uncovered) city c *opens* facility f.

Let Y be the random variable that denotes the set of facilities that A opens. We will show that the probability Y covers all cities **and** $|Y| \leq \frac{n}{2} - d\sqrt{n}$ (for constant d) is very small.

Lemma 3. $\Pr(Y$ *covers all cities and* $|Y| \leq \frac{n}{2} - d\sqrt{n}) \leq 4e^{-\frac{2}{3}}$.

Lemma 3 suggests that $E[A] \geq (1 - 4e^{-\frac{2}{3}}) \cdot 2 \cdot (\frac{n}{2} - d\sqrt{n}) + n$. Observe that for arbitrarily small ϵ, one can find a constant d such that $\frac{\mathbf{E}[A]}{\mathbf{E}[OPT]} \geq 2 - \epsilon$. □

5 Randomized Priority Makespan Scheduling

The results we show in this Section hold under the assumption that the algorithm does not know the number of jobs in the input (the lower bounds for deterministic priority algorithms for the problem, shown in [8], rely on this assumption as well).

5.1 Fully-Randomized Priority Makespan Scheduling

Theorem 4. *No fully-randomized priority makespan scheduling algorithm has approximation ratio better than* $\frac{12}{11}$*, for all* $m \geq 2$*.*

Proof. As in the proof of the deterministic $\frac{7}{6}$ bound in [8], the set S of potential jobs consists of $2\lfloor \frac{m}{2} \rfloor$ jobs of size 3, and $3\lceil \frac{m}{2} \rceil$ jobs of size 2. If the actual input is S itself, the optimal makespan is 6, and is achieved by scheduling 2 jobs of size 3 on each of $\lfloor \frac{m}{2} \rfloor$ machines and 3 jobs of size 2 on each of the remaining $\lceil \frac{m}{2} \rceil$ machines. The following fact is shown in [8]:

Fact 1 *If the actual input set is* S and *the algorithm schedules two jobs of different sizes in the same machine, then the makespan of the algorithm is at least 7.*

We will bound the expected makespan of a deterministic algorithm A for a probability distribution over the actual inputs defined as follows: with probability p, to be determined later, the actual input to A is S, while with probability $1-p$ the actual input is a set $S' \subset S$, with $|S'| = m + \lceil \frac{m}{2} \rceil + 1$ (S' can be any subset of S of the above size).

Lemma 4. *At the point where* A *has scheduled precisely* $m + \lceil \frac{m}{2} \rceil + 1$ *jobs, one of the following holds:*

- *The current makespan is at least 6.*
- *Two jobs of different sizes were scheduled on the same machine.*
- *No two jobs of different sizes were scheduled on the same machine, and if* l_2,l_3 *is the number of machines on which jobs of sizes 2 and 3 were scheduled, respectively, then* $l_2 > \lceil \frac{m}{2} \rceil$ *or* $l_3 > \lfloor \frac{m}{2} \rfloor$*.*

Denote the optimal makespan on input S' by $opt(S')$. It is not difficult to see that $opt(S') \in \{4, 5\}$. Note that $\mathbf{E}[OPT] = 6p + opt(S')(1-p)$. To bound $\mathbf{E}[A]$ we consider cases concerning the behavior of the algorithm at the point where exactly $m + \lceil \frac{m}{2} \rceil + 1$ jobs have been scheduled, as suggested by Lemma 4.

- The (current) makespan of the algorithm is at least 6. Then $\mathbf{E}[A] \geq 6p + 6(1-p) = 6$.
- Two jobs of different sizes are scheduled on the same machine. Then $\mathbf{E}[A] \geq 7p + 5(1-p)$ (from Fact 1).
- No two jobs of different size were scheduled on the same machine, and $l_2 > \lceil \frac{m}{2} \rceil$ or $l_3 > \lfloor \frac{m}{2} \rfloor$, or, equivalently, $l_3 < \lfloor \frac{m}{2} \rfloor$ or $l_2 < \lceil \frac{m}{2} \rceil$. On the event the actual input is S, this implies that either two jobs in S of different sizes are eventually scheduled on the same machine, or one of the following holds: A machine will be assigned at least 4 jobs of size 2, or a machine will be assigned at least 3 jobs of size 3 (depending on whether $l_3 > \lfloor \frac{m}{2} \rfloor$, or $l_2 > \lceil \frac{m}{2} \rceil$, respectively). In each case, the makespan of A is at least 7. Thus, $\mathbf{E}[A] \geq 7p + opt(S')(1-p)$.

It follows that $\max_p \frac{\min(7p+opt(S')(1-p),6)}{6p+opt(S')(1-p)}$, with the constraint $opt(S') \in \{4, 5\}$, is a lower bound on the approximation ratio of A. The expression is maximized at $p = \frac{1}{2}$, which gives a lower bound of $\frac{12}{11}$. □

5.2 Fixed-Priority Makespan Scheduling with Randomized Decisions

Theorem 5. *No* Fixed Priority *randomized-decisions algorithm for makespan scheduling with $m \geq 3$ has approximation ratio better than $\frac{10}{9}$.*

Proof sketch. The set of potential jobs S is as defined in the proof of Theorem 4. In order to determine the appropriate input distribution, we observe the sequence σ of the first m jobs in the ordering specified by a given deterministic Fixed Priority algorithm A, and consider cases for σ.

6 Oblivious vs. Adaptive Adversaries

What if the adversary is not oblivious, but instead it can observe the outcome of the algorithm's random orderings and/or random decisions? We call such an adversary *adaptive*[5] using, again, terminology from the analysis of online algorithms. Since the adaptive adversary has knowledge of the algorithm's random choices, it works by removing input inputs from the set of potential input items based on the outcome of such choices. This means that the actual input to the algorithm, and hence the value of the optimal solution will both be random variables. A priority algorithm is a c-approximation against an adaptive adversary if $\mathbf{E}[\mathcal{A}(\sigma)] \leq c \cdot \mathbf{E}[OPT(\sigma)]$.

The following theorem provides some evidence that in the context of priority algorithms, the adaptive adversary may be too powerful to be of any interest (reminiscent of the online world, where there is no advantage to using randomization against *adaptive-offline* adversaries [5]).

Theorem 6. *Suppose that, using the techniques of [8], we can show (constructively) that a problem is not c-approximable by a deterministic priority algorithm. Then the problem is not c-approximable by any randomized priority algorithm against the adaptive adversary.*

7 Open Questions and Future Directions

Apart from improving the lower bounds, if possible, it would be very interesting to use the intuition behind the lower-bound constructions so as to design better randomized priority algorithms. For instance, would our results suggest better upper bounds for randomized priority makespan scheduling?

We believe that the framework is applicable to a wider class of optimization problems; in particular, we believe that it can be applied in the model of Impagliazzo and Davis [19], so as to address the power of randomized greedy-like algorithms for graph-related problems. Finally, several questions that have been asked in the context of the competitive analysis of online algorithms can be asked about the class of algorithms we study. For instance, how critical is memory in

[5] Not to be confused with adaptive priority algorithms.

randomized priority algorithms? (see [6] for a discussion of the situation in the online world). Also, can we extend Theorem 6 to show that if a randomized priority algorithm is a c-approximation against the adaptive adversary, then there exists a deterministic c-approximation priority algorithm?

Acknowledgments

The author would like to thank Allan Borodin for several helpful discussions and comments, as well as Travis Gagie, Avner Magen, and Tasos Viglas for their suggestions.

References

1. S. Albers. On randomized online scheduling. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computation*, pages 134–143, 2002.
2. S. Angelopoulos and A. Borodin. On the power of priority algorithms for facility location and set cover. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 26–39, 2002.
3. A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating throughput in real-time scheduling. *SIAM Journal of Computing*, 31(2):331–352, 2001.
4. Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51(3):359–366, 1995.
5. S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. *Algorithmica*, 11(1):2–14, 1994.
6. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
7. A. Borodin, N. Linial, and M.E. Saks. An optimal algorithm for metrical task systems. *Journal of the ACM*, 39(4):745–763, 1992.
8. A. Borodin, M. Nielsen, and C. Rackoff. (Incremental) priority algorithms. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 752–761, 2002.
9. B. Chen, A. van Vliet, and G. J. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Processing Letters*, 51:219–222, 1994.
10. M. Chrobak, L. Larmore, N. Reingold, and J. Westbrook. A better lower bound on the competitive ratio of the randomized 2-server problem. *Information Processing Letters*, 63:79–83, 1997.
11. R. Fleischer and M. Wahl. Online scheduling revisited. In *Proceedings of the 8th Annual European Symposium on Algorithms*, pages 202–210, 2000.
12. D. Fotakis. On the competitive ratio for online facility location. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming*, pages 637–652, 2003.
13. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, NY, 2nd edition, 1983.
14. T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 564–565, 2000.

15. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell Sys. Tech. J.*, 45:1563–1581, 1966.
16. R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.
17. S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 649–657, 1998.
18. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.
19. R. Impagliazzo and S. Davis. Models of greedy algorithms for graph problems. In *Proceedings of the 15th Symposium on Discrete Algorithms*, 2004. To appear.
20. K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computation*, pages 731–740, 2002.
21. M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. A greedy facility location algorithm analyzed using dual fitting. In *Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 127–137, 2001.
22. M. Mahdian, J. Ye, and J. Zhang. A 1.52-approximation algorithm for the uncapacitated facility location problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 229–242, 2002.
23. R. R. Mettu and C. G. Plaxton. The online median problem. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 339–348, 2000.
24. A. Meyerson. Online facility location. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 426–431, 2001.
25. S. K. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23(1):116–127, 1976.
26. S. Seiden, J. Sgall, and G. J. Woeginger. Semi-online scheduling with decreasing job sizes. *Operations Research Letters*, 27(5):215–221, 2000.
27. J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63:51–55, 1997.
28. J. Sgall. On-line scheduling–a survey. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of the Art*, Lecture Notes in Computer Science, pages 196–231. Springer Verlag, 1998.
29. D.B. Shmoys. Approximation algorithms for facility location problems. In K. Jansen and S. Khuller, editors, *Approximation Algorithms for Combin. Optimization*, volume 1913 of *LNCS*. Springer, 2000.
30. D.B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.

Tradeoffs in Worst-Case Equilibria

Baruch Awerbuch[1], Yossi Azar[2,⋆], Yossi Richter[2,⋆⋆], and Dekel Tsur[2]

[1] Dept. of Computer Science, Johns Hopkins University, Baltimore, MD, 21218
baruch@cs.jhu.edu
[2] School of Computer Science, Tel Aviv University, Tel Aviv, 69978, Israel
{azar,yo,dekelts}@tau.ac.il

Abstract. We investigate the problem of routing traffic through a congested network in an environment of non-cooperative users. We use the worst-case coordination ratio suggested by Koutsoupias and Papadimitriou to measure the performance degradation due to the lack of a centralized traffic regulating authority. We provide a full characterization of the worst-case coordination ratio in the restricted assignment and unrelated parallel links models. In particular, we quantify the tradeoff between the "negligibility" of the traffic controlled by each user and the coordination ratio. We analyze both pure and mixed strategies systems and identify the range where their performance is similar.

1 Introduction

Overview: In most communication networks it is infeasible to maintain one centralized authority to route traffic efficiently. As a result, users may decide individually how to route their traffic. Each user behaves selfishly in the sense that he wishes to minimize his transfer cost while being aware of the network congestion caused by other users. A system of users decisions is said to be in a Nash equilibrium if no user can benefit from changing his decision. Simple examples in game theory show that the performance of systems in Nash equilibrium, achieved by non-cooperative users, can be far from the global optimum. Recently, the question of quantifying the decrease in network performance caused by the lack of a centralized authority, received considerable attention among researchers. Koutsoupias and Papadimitriou [5, 8] suggested to investigate the worst-case coordination ratio, which is the ratio between the worst possible Nash equilibrium and the global optimum, as a mean to understand the cost incurred due to the lack of a centralized regulating authority.

Problems of this type have been studied lately by two approaches. The worst-case coordination ratio in a network composed of m parallel related links was analyzed in [4, 5, 8], while making no assumptions on the relative amount of traffic controlled by each user. On the other hand, [11, 12] obtained improved bounds

⋆ Research supported in part by the Israeli Ministry of industry and trade and by the Israel Science Foundation.
⋆⋆ Research supported in part by the Israeli Ministry of industry and trade.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 41–52, 2004.
© Springer-Verlag Berlin Heidelberg 2004

for general networks while assuming that each user controls only a negligible fraction of the total traffic. We attempt to bridge between these two approaches, by analyzing the worst-case coordination ratio as a function of the relative fraction of the total traffic controlled by each user. Specifically, we quantify the required "negligibility" of each user's traffic, needed to bound the worst-case coordination ratio by a constant.

We focus on two new models. First, we study the restricted assignment model (also called the subset model) which is defined as follows. The network consists of m parallel links, there are n users where user j has amount of traffic w_j, that can be transferred through any link from a subset S_j of the m links. We consider both the *pure strategies* case where each user selects one link to transfer his traffic, and the more general case of *mixed strategies* where each user decides a probability distribution over his allowable links. In both cases each user is aware of the decisions made by other users. Each user behaves selfishly and wishes to minimize his cost by assigning his traffic to the least loaded link. The global objective however, is to minimize the load of the most loaded link. We note that even though this model is a simplification of real communication networks, it captures the essence of basic networking problems as pointed out by [5, 6, 8].

Additionally, we study the more general model of unrelated links in which a task j ($j = 1, \ldots, n$) is associated with an m-vector $\boldsymbol{w}_j$ specifying its weight on each link. We analyze the worst-case coordination ratio as a function of the maximum stretch s in the system, where $s = \max_{j,i,l:w} <\infty \frac{w}{w}$.

Our results: We provide a full characterization of the coordination ratio for both pure and mixed strategies in the restricted assignment and unrelated links models. Specifically, we prove the following results:

- We show the following tight bounds for the restricted assignment model, as a function of the ratio r between the optimal assignment and the largest task:
  - For $1 \leq r \leq \log m$ we prove that the coordination ratio is bounded by $\Theta\left(\frac{\log m}{r \cdot \log(1+\frac{\log}{})}\right)$ in the pure strategies case, and by $\Theta\left(\frac{\log m}{r \cdot \log\log(1+\frac{\log}{})}\right)$ in the mixed strategies case.
  - For $r = \Omega(\frac{\log m}{\epsilon^2})$ we show that the coordination ratio for both pure and mixed strategies systems is bounded by $1 + \epsilon$.

  Note that general bounds of $\Theta(\frac{\log m}{\log\log m})$ for pure strategies and $\Theta(\frac{\log m}{\log\log\log m})$ for mixed strategies are obtained when $r = 1$, i.e. when making no assumptions on the largest task in the system. We also note that for $r = \Omega(\log m)$ the coordination ratio is bounded by a constant, for both pure and mixed strategies.
- In the unrelated links model we prove that the coordination ratio is bounded by $\Theta\left(s + \frac{\log m}{\log(1+\frac{\log}{})}\right)$ in pure strategies systems, and $\Theta\left(s + \frac{s \cdot \log m}{\log(1+s \cdot \log\frac{\log}{})}\right)$ when mixed strategies are allowed.

Some of our results are obtained by extending the techniques used in [4] to the restricted assignment and unrelated links models.

Related work: Koutsoupias and Papadimitriou [5] initiated the study of worst-case coordination ratio in networks composed of m-parallel related links with possibly different speeds. They first investigated the case of two links and proved a worst-case coordination ratio of 3/2 for the case of identical links, and $\phi = \frac{1+\sqrt{5}}{2}$ for links with possibly different speeds. They also obtained non-tight bounds for the general case. Mavronicolas and Spirakis [6] continued this line of research while focusing on the special case of fully-mixed strategies in which the probability of assigning any task to any link is non-zero. They proved that in this case the worst-case coordination ratio is bounded by $\Theta(\frac{\log m}{\log \log m})$ for both the identical links model and the general related links model where all tasks have equal weights and $m \leq n$. Czumaj and Vöcking [4] proved tight bounds for the m-parallel related links model, and showed that the worst-case coordination ratio is bounded by $\Theta(\frac{\log m}{\log \log m})$ in the identical links model and by $\Theta(\frac{\log m}{\log \log \log m})$ in the general related links model. Czumaj *et al.* [3] continued to study this problem and characterized the coordination and bicriteria ratios for different families of cost functions.

Roughgarden *et al.* [10, 12] also examined the degradation in network performance due to unregulated traffic. Their model deals with general networks where users adopt *pure strategies* only, and the amount of traffic of each user is assumed to be a negligible fraction of the total traffic. The objective is to minimize the total latency. They proved that when the latency of each edge is a linear function of the edge congestion, any flow at Nash equilibrium has total latency at most 4/3 of the optimal flow. Although for general latency functions the coordination ratio is unbounded, the following bicriteria result can be shown: for any network with continuous non-decreasing latency functions, a flow at Nash equilibrium has total latency no more than that of an optimal flow forced to route twice as much traffic. Roughgarden [11] also showed that the cost of unregulated traffic does not depend on the complexity of the network topology. He also studied the impact of latency functions belonging to specific classes. Roughgarden *et al.* [1, 2, 9] studied various ways to construct and price networks such that the cost incurred in unregulated traffic is minimized.

Paper structure: The paper is organized as follows. Section 2 includes formal definitions and notations. The restricted assignment model is studied in section 3. In section 4 we analyze the unrelated links model.

2 Definitions and Notations

The restricted assignment model is defined as follows: there are m parallel links and n users, where user j $(j = 1, \ldots, n)$ has a task with weight w_j, that can be assigned to any link from a subset S_j of the m links. We denote the largest task in the system by $w_{max} = \max_{1 \leq j \leq n} w_j$. Given an instance of the problem we define the *global optimum* (denote it by OPT) to be the assignment of tasks to links that minimizes the maximum load of a link. We denote the ratio between the value of the optimal solution and the largest task by $r = \frac{OPT}{w_{max}}$. The unrelated

links model is more general. Task j ($j = 1, \ldots, n$) is associated with an m-vector $\boldsymbol{w}_j$, where w_{ij} indicates the weight of task j on link i.

We assume that the users are non-cooperative and each one wishes to minimize his own cost with no regard to the global optimum. We consider two types of users strategies systems:

1. **Pure strategies:** user j selects link $l_j \in S_j$ and assigns his task to it. Each user is aware of the choices made by all other users when making his decision.
2. **Mixed strategies:** user j selects a probability distribution $\{p_{ij}\}$ ($i \in S_j$) over the allowable set of links for task j. Each user is aware of the probability distributions selected by all other users.

For the remainder of this section we regard pure strategies as a special case of mixed strategies, and describe our definitions in terms of mixed strategies systems. Given a system S of mixed strategies with probability distributions $\{p_{ij}^S\}$, we define the following random variables:

- A set of indicator random variables $\{X_{ij}^S\}$, where X_{ij}^S indicates whether task j is assigned to link i. By definition: $\Pr[X_{ij}^S = 1] = p_{ij}^S$.
- For each link i ($i = 1, \ldots, m$) we define a random variable L_i^S, indicating the total load on the link: $L_i^S = \sum_{j=1}^{n} w_j \cdot X_{ij}^S$. We denote the maximum expected load by $\mu^S = \max_{1 \le i \le m} E[L_i^S]$.
- We define a random variable $L_{max}^S = \max_{1 \le i \le m} L_i^S$ to indicate the maximum link load, and denote its expectancy by $\mu_{max}^S = E[L_{max}^S]$. Clearly, $\mu_{max}^S \ge \mu^S$.

For simplicity of notation, throughout the paper we omit the superscript S when meaning is clear from context.

Definition 1. *The* expected cost *of user j for assigning his task to link i in system S is defined as:* $c_{ij}^S = E[L_i^S] + (1 - p_{ij}^S)w_j$.

Definition 2. *A system S is said to be in* Nash Equilibrium *if and only if for every task j and link i, $p_{ij}^S > 0$ only if $i = \arg\min_{1 \le k \le m} c_{kj}^S$.*

Definition 3. *The* worst-case coordination ratio *of an instance of the problem is defined as $R = \max_S \frac{\mu_{max}}{OPT}$, where the maximum is taken over all strategies systems S in Nash equilibrium.*

3 Worst-Case Equilibria in the Restricted Assignment Model

In this section we provide tight bounds for worst-case equilibria in the restricted assignment model. We investigate both pure and mixed strategies. We first show upper bounds for the problem, and then provide matching lower bounds.

3.1 Upper Bounds for Restricted Assignment

We begin by proving an upper bound on the maximum expected load in any system in Nash equilibrium. Recall that $r = \frac{OPT}{w_{max}}$, without loss of generality we assume that r is integral.

Lemma 1. *Let S be any system in Nash equilibrium. Then $\mu^S \leq (\beta + \frac{1}{r}) \cdot OPT$, where β satisfies the inequality: $e(\frac{\beta}{e})^\beta \leq m^{\frac{1}{r}}$.*

Proof. We order the links by non-increasing order of their expected loads. For every $k \geq 1$, define m_k to be the minimal integer such that $E[L_{m_k+1}] < k \cdot w_{max}$ ($m_k = m$ if there is no such integer). We define $l = \lfloor \frac{\mu}{w_{max}} \rfloor$. Note that $m_l \geq 1$. The next claim states an important property regarding the relation between m_k and m_{k+1}.

Claim 1. *For every $k \geq 1$, $\frac{m}{m_{k+1}} \geq \frac{k+1}{r}$.*

Proof. Denote by J_{k+1} the set of tasks with positive probability to be assigned to any link from $[1, \ldots, m_{k+1}]$, and denote the total weight of tasks from J_{k+1} by $W(J_{k+1}) = \sum_{j \in J_{k+1}} w_j$. Consider the way OPT schedules the tasks from J_{k+1}. OPT can not assign a task from J_{k+1} to a link with index larger than m_k. Let us assume, for contradiction, that OPT assigns task $j \in J_{k+1}$ to link $t > m_k$. In addition, we assume that in our system $p_{qj} > 0$ for some link $q \leq m_{k+1}$. There follows:

$$\begin{aligned} c_{qj} &= E[L_q] + (1 - p_{qj})w_j \geq (k+1)w_{max} + (1 - p_{qj})w_j \geq (k+1)w_{max} \\ &\geq k \cdot w_{max} + (1 - p_{tj})w_j > c_{tj}, \end{aligned}$$

where the first inequality follows from the definition of q and the third inequality results from $w_j \leq w_{max}$ $(j = 1, \ldots, n)$. This contradicts the fact that the system S is in Nash equilibrium. Hence, OPT must assign all tasks from J_{k+1} to links in the range $[1, \ldots, m_k]$. Let L_i^{OPT} denote the load on link i in the optimal assignment. Recall that $w_{max} = \frac{OPT}{r}$. We conclude that

$$m_k \cdot OPT \geq \sum_{i=1}^{m} L_i^{OPT} \geq W(J_{k+1}) \geq \sum_{i=1}^{m_{k+1}} E[L_i] \geq (k+1) \left(\frac{OPT}{r} \right) m_{k+1},$$

which yields the desired inequality. □

We use Claim 1 iteratively together with the inequality $(n!/k!) \geq \frac{(n/e)}{(k/e)}$ to obtain the following:

$$m \geq m_r \geq \frac{l(l-1)\cdots(r+1)}{r^{l-r}} = \frac{l!}{r! \cdot r^{l-r}} \geq \frac{(\frac{l}{e})^l}{(\frac{r}{e})^r \cdot r^{l-r}} \ .$$

Substituting $l = \beta \cdot r$ we get: $e(\frac{\beta}{e})^\beta \leq m^{1/r}$ and $\mu^S \leq (l+1)\frac{OPT}{r} = (\beta + \frac{1}{r})OPT$. □

Pure Strategies. In the following theorem, derived from Lemma 1, we show an upper bound on the worst-case coordination ratio as a function of the ratio r between the optimal solution and the largest task in the system.

Theorem 1. *For pure strategies, when* $1 \leq r \leq \log m$, $R = O\left(\frac{\log m}{r \log(1+\frac{\log}{r})}\right)$.

As a direct result from Theorem 1, we obtain the following general upper bounds for two extreme cases. In the first case we make no assumptions regarding the amount of traffic controlled by each user, i.e. $r = 1$. In the second case the tasks are relatively small, i.e. $r = \Theta(\log m)$.

Corollary 1. *For pure strategies,* $R = O(\frac{\log m}{\log \log m})$.

Corollary 2. *For pure strategies, when* $r = \Theta(\log m)$, *we have* $R = O(1)$.

Next, we study the coordination ratio in networks where each user controls a small fraction of the total traffic, i.e. $r = \Omega(\log m)$. The next theorem shows that in such networks the coordination ratio is close to 1.

Theorem 2. *For any* $0 < \epsilon < 1$, *if* $r = \Omega(\frac{\log m}{\epsilon^2})$, *then* $R \leq 1 + \epsilon$.

Proof. From Lemma 1 we know that $R \leq \beta + \frac{1}{r}$ where β satisfies the inequality: $e(\frac{\beta}{e})^{\beta} \leq m^{1/r}$. When taking $r = \Omega(\frac{\log m}{\epsilon^2})$ we obtain using Taylor expansion:

$$e^{O(\epsilon^2)} \geq m^{1/r} \geq e\left(\frac{\beta}{e}\right)^{\beta} = e^{\frac{1}{2}(\beta-1)^2 - O((\beta-1)^3)},$$

and the inequality yields $R \leq \beta + \frac{1}{r} \leq 1 + \epsilon$. □

Mixed Strategies. The following theorem gives an upper bound on the coordination ratio for the mixed strategies case.

Theorem 3. *For mixed strategies,* $R = O\left(\frac{\log m}{r \cdot \log \log(1+\frac{\log}{r})}\right)$.

Proof. For convenience we scale the tasks weights such that $w_{max} = 1$. Consider an arbitrary link i. Recall that $L_i = \sum_{j=1}^{n} w_j X_{ij}$, where $w_j \leq w_{max} = 1$ for $j = 1, \ldots, n$, and $\mu = \max_{1 \leq i \leq m} E[L_i]$. We apply Hoeffding inequality and obtain for every $c > 1$:

$$\Pr[L_i \geq c \cdot \mu] \leq \left(\frac{e \cdot E[L_i]}{c \cdot \mu}\right)^{c \cdot \mu} \leq \left(\frac{e \cdot \mu}{c \cdot \mu}\right)^{c \cdot \mu} = \left(\frac{e}{c}\right)^{c \cdot \mu}.$$

We can now apply the union bound to obtain: $\Pr[L_{max} \geq c \cdot \mu] \leq m(e/c)^{c \cdot \mu}$. For every integer $\alpha > 1$ we can upper bound the expected maximum load by

$$\begin{aligned} \mu_{max} &\leq \alpha \cdot \mu + \mu \cdot \sum_{k=\alpha}^{\infty} \Pr[L_{max} \geq k \cdot \mu] \\ &\leq \alpha \cdot \mu + m \cdot \mu \sum_{k=\alpha}^{\infty} (e/k)^{k \cdot \mu} \leq \alpha \cdot \mu + 2m \cdot \mu \cdot (e/\alpha)^{\alpha \cdot \mu}, \end{aligned}$$

where the last inequality follows from the fact that the sequence is sub-geometric. We can substitute α for $\Gamma^{-1}(m^{1/\mu})$, where Γ^{-1} is the inverse Gamma function and $\Gamma^{-1}(x) = \Theta(\frac{\log x}{\log\log x})$. We then have: $(e/\alpha)^{\alpha\cdot\mu} \leq 1/m$, hence $\mu_{max} \leq (\alpha + 2)\mu$. Since $\alpha = O\left(\frac{\log m}{\mu\cdot\log\frac{\log m}{\mu}}\right)$ there follows: $\mu_{max} = O\left(\frac{\log m}{\log\frac{\log m}{\mu}}\right)$. Since $OPT \geq r$, Theorem 1 implies that $\mu = O\left(\frac{\log m}{\log(1+\frac{\log m}{r\mu})}\right)$, substituting μ we conclude that $\mu_{max} = O\left(\frac{\log m}{\log\log(1+\frac{\log m}{r\mu})}\right)$ and therefore $R = O\left(\frac{\log m}{r\cdot\log\log(1+\frac{\log m}{r\mu})}\right)$. □

The following upper bounds for two extreme cases are derived directly from Theorem 3.

Corollary 3. *In the restricted assignment problem* $R = O(\frac{\log m}{\log\log\log m})$.

Corollary 4. *When* $r = \Theta(\log m)$ *we have* $R = O(1)$.

The next theorem refers to networks where each user controls a small fraction of the total traffic.

Theorem 4. *For any* $0 < \epsilon < 1$, *if* $r = \Omega(\frac{\log m}{\epsilon^2})$, *then* $R \leq 1 + \epsilon$.

Proof. For convenience we scale the tasks weights such that $w_{max} = 1$, hence $OPT = r$. By the Chernoff bound, for every $1 \leq i \leq m$, and $0 < \epsilon < 1$:

$$\begin{aligned}\Pr[L_i \geq (1+\epsilon/3)\mu] &\leq \Pr[\ L_i + (\mu - E[L_i])\ \geq\ (1+\epsilon/3)\mu\ ] \\ &\leq e^{-\frac{2}{27}} \leq m^{-3},\end{aligned}$$

where the first inequality follows since $\mu - E[L_i] \geq 0$, the second inequality results from the fact that $E[\ L_i + (\mu - E[L_i])\ ] = \mu$, and the last inequality follows from $\mu \geq OPT = r$. By applying the union bound, with high probability $L_{max} \leq (1+\epsilon/3)\mu$ and therefore $\mu_{max} \leq (1+\epsilon/3)\mu$. By using Theorem 2 with $\epsilon/3$ we conclude that $\mu_{max} \leq (1+\epsilon/3)(1+\epsilon/3)OPT \leq (1+\epsilon)OPT$ and the theorem follows. □

3.2 Lower Bounds for Restricted Assignment

We begin by proving a tight lower bound for the pure strategies case. We note that our lower bounds are proved even for unit tasks.

Theorem 5. *For pure strategies,* $R = \Omega\left(\frac{\log m}{r\cdot\log(1+\frac{\log m}{r})}\right)$.

Proof. We construct the following problem instance:
links: we partition the m links into $l+1$ groups (the value of l will be determined later) such that in group $k = 0, \ldots, l$ there are $\sqrt{m}\left\lfloor\frac{l!}{r}\cdot\frac{r}{k!}\right\rfloor$ links. Denote the number of links in group k by n_k.
tasks: we partition the tasks into l groups. In group $k = 1, \ldots, l$ there are $k\cdot n_k$ unit tasks, each can be assigned to any link from groups $[k-1, \ldots, l]$.

Observe that $OPT \leq r+1$ for this problem instance. The optimal solution assigns the tasks of group k $(k = 1, \ldots, l)$ to the links in group $k-1$, at most $r+1$ tasks per link. We define the following system of pure strategies, denote it by S: all tasks from group k $(k = 1, \ldots, l)$ are assigned to links from group k, k tasks per link.

Claim 2. *The system S is in Nash equilibrium.*

Proof. Denote by S_k $(k = 1, \ldots, l)$ the set of links to which tasks from task group k can be assigned. Let j be a task from group k and consider the assignment of j to link i from link group k. Clearly, $c_{ij} = k$, and for each link $t \in S_k$ we have $c_{tj} \geq (k-1)+1 = k \geq c_{ij}$. Hence the system S is in Nash equilibrium. □

We now turn to bound the coordination ratio. Since the number of links is m we should satisfy:

$$m \geq \sqrt{m} \cdot \frac{l!}{r^l} \sum_{k=1}^{l} \frac{r^k}{k!} = \sqrt{m} \cdot \frac{l!}{r^l} \cdot e^r,$$

using Stirling's formula we bound the last expression from above by

$$\sqrt{m} \cdot \frac{(1+o(1))\sqrt{2\pi l} \cdot (\frac{l}{e})^l}{r^l} \cdot e^r \leq m,$$

and by taking $l = \beta r$, we conclude that $R = \Omega(\frac{l}{r}) = \Omega(\beta) = \Omega\left(\frac{\log m}{r \log(1+\frac{\log}{\ })}\right)$. □

Using Theorem 5, we can derive a tight lower bound (for pure and mixed strategies) for the case where each user controls a small fraction of the total traffic. The proof is omitted.

Theorem 6. *For any $0 < \epsilon < 1$, if $r = O(\frac{\log m}{\epsilon^2})$, then $R \geq 1+\epsilon$.*

We can modify our construction from Theorem 5 to obtain a tight lower bound for the mixed strategies case.

Theorem 7. *For mixed strategies, $R = \Omega\left(\frac{\log m}{r \cdot \log\log(1+\frac{\log}{\ })}\right)$.*

Proof. We slightly modify the problem instance constructed in the proof of Theorem 5. Everything remains the same except tasks group l now contains $(l-1) \cdot n_l$ tasks. Clearly, $OPT \leq r+1$. We introduce the following system of mixed strategies, denote it by S:

- All tasks from group k $(k = 1, \ldots, l-1)$ are assigned to links from group k, k tasks per link (the same as the construction in the proof of Theorem 5).
- Each task from group l has uniform distribution over the links from group l.

We first prove that the system S is in Nash equilibrium.

Claim 3. *The system S is in Nash equilibrium.*

Proof. The cost of task j from group l on any link i in group l is: $c_{ij} = (l-1) + (1-1/\sqrt{m})$. On the other hand, for any link t not in group l: $c_{tj} = (l-1)+1 > c_{ij}$. The proof concerning tasks from groups $[1, \ldots, l-1]$ is identical to the proof given in Claim 2. □

In the following claim we determine a lower bound on μ^S_{max}.

Claim 4. $\mu^S_{max} = \Omega\left(\frac{\log m}{\log\log(1+\frac{\log}{})}\right)$.

Proof. Consider the assignment of tasks to links in group l. There are $(l-1)\cdot\sqrt{m}$ unit tasks each with uniform distribution over the $\sqrt{m}$ links. This corresponds to a model of throwing $(l-1)\cdot\sqrt{m}$ balls uniformly to $\sqrt{m}$ bins (see e.g. [7]). In this model the expected maximum occupancy is: $\Omega\left(l + \frac{\log m}{\log((\log m)/l)}\right)$. In our case this lower bound corresponds to $\mu^S_{max} = \Omega\left(\frac{\log m}{\log\log(1+\frac{\log}{})}\right)$. □

Since $OPT \leq r+1$, our lower bound on the coordination ratio in the case of mixed strategies follows directly from Claim 4. □

4 Analysis of the Unrelated Links Model

Recall that in the unrelated links model, a task j is associated with an m-vector $\boldsymbol{w}_j = (w_{1j}, \ldots, w_{mj})$ specifying its weight on each link. Clearly, this model generalizes the restricted assignment model. We define the maximum stretch in the system as $s = \max_{j,i,l:w\ <\infty} \frac{w}{w}$. In the next sections we show tight bounds for pure and mixed strategies as a function of s. Note that by increasing s the coordination ratio becomes arbitrarily large.

4.1 Upper Bounds for the Unrelated Link Model

We begin by proving an upper bound on the maximum expected load in any system in Nash equilibrium.

Lemma 2. *Let S be a system in Nash equilibrium. Then*

$$\mu^S = O\left(s + \frac{\log m}{\log(1+\frac{\log m}{s})}\right) \cdot OPT.$$

Proof. We order the links by non-increasing order of their expected loads. For every $k \geq 1$, define m_k to be the minimal integer such that $E[L_{m\ +1}] < k \cdot OPT$ ($m_k = m$ if there is no such integer). Let $h = \lfloor \mu^S / OPT \rfloor$. The following claim shows a relation between m_k and m_{k+1}.

Claim 5. *For every $k \geq 1$, $\frac{m}{m\ +1} \geq \frac{k+1}{s}$.*

Proof. Denote by J_{k+1} the set of tasks assigned by S with positive probability to a link from $[1,\ldots,m_{k+1}]$, and $W(J_{k+1})$ is the total weight of the tasks from J_{k+1} under the assignment of S. Suppose that there is a task $j \in J_{k+1}$ which is assigned by OPT to a link $t > m_k$, and let $q \leq m_{k+1}$ be a link to which j is assigned by S with positive probability. Then,

$$c_{qj} \geq (k+1)\cdot OPT \geq k \cdot OPT + (1-p_{tj})w_{tj} > c_{tj},$$

contradicting the assumption that S is in Nash equilibrium. Therefore, all the tasks in J_{k+1} are assigned by OPT to links from $[1,\ldots,m_k]$. Hence,

$$m_k \cdot OPT \geq \sum_{i=1}^{m} L_i^{OPT} \geq \frac{W(J_{k+1})}{s} \geq \frac{(k+1)\cdot OPT}{s}\cdot m_{k+1},$$

and the claim follows. □

Now, if $h \leq s$ then we are done. Otherwise, using Claim 5 we obtain that

$$m \geq m_s \geq \frac{h(h-1)\cdots(s+1)}{s^{h-s}} = \frac{h!}{s!\cdot s^{h-s}} \geq \left(\frac{l}{e^{1-s/h}s}\right)^h.$$

It follows that $h = O\left(\frac{\log m}{\log(1+\frac{\log}{\ })}\right)$. □

The next two theorems bound the worst-case coordination ratio for pure and mixed strategies. Theorem 8 follows directly from Lemma 2. The proof of Theorem 9 is by a similar analysis to the one used in the proof of Theorem 3, and is omitted.

Theorem 8. *For pure strategies* $R = O\left(s + \frac{\log m}{\log(1+\frac{\log}{\ })}\right)$.

Theorem 9. *For mixed strategies* $R = O\left(s + \frac{s\cdot\log m}{\log(1+s\cdot\log\frac{\log}{\ })}\right)$.

4.2 Lower Bounds for the Unrelated Link Model

In this section we show matching lower bounds.

Theorem 10. *For pure strategies,* $R = \Omega\left(s + \frac{\log m}{\log(1+\frac{\log}{\ })}\right)$.

Proof. We prove the theorem by constructing two instances of the problem. Without loss of generality we assume that s is integral.

We first assume that $s \leq \log m$. We give a construction similar to the one in Theorem 5: We partition the links into $Ks+1$ groups, where $K \leq \Gamma^{-1}(m^{\frac{1}{3}})$. For $k = 0,\ldots,K-1$ and $i = 1,\ldots,s$, group number $ks+i$ contains $n_{ks+i} = \sqrt{m}\cdot\frac{(K!)}{(k!)\ (k+1)}$ links, and group 0 contains $n_0 = m - \sum_{l=1}^{ks+1} n_l$ links. Note that

$$\sqrt{m}\cdot(K!)^s + \sum_{l=1}^{ks+1} n_l \le \sqrt{m}\cdot(K!)^s + \sqrt{m}\cdot\sum_{k=0}^{K-1} s\frac{(K!)^s}{(k!)^s} \le \sqrt{m}\cdot(K!)^s\,(1+es) \le m,$$

so $n_0 \ge \sqrt{m}\cdot(K!)^s$.

The tasks are partitioned into Ks groups. For $k = 0,\ldots,K-1$ and $i = 1,\ldots,s$, the $(ks+i)$-th group contains $(k+1)\cdot n_{ks+i}$ tasks. Each task in that group has weight 1 on a link from group $ks+i-1$, weight $s-(s-i)/(k+1)$ on a link from group $ks+i$, and an infinite weight on all other links.

The optimal assignment is to assign each task from the l-th group of tasks to a distinct link in the $(l-1)$-th group of links. Thus, $OPT = 1$. Now, consider the following system S of pure strategies: Each task from the l-th group is assigned to a distinct link in the l-th group of links. Clearly, the load on a link from the l-th group is exactly l for $l \ge 1$, and in particular, the maximum load is Ks.

The system S is in Nash equilibrium: For a task j from the l-th group that was assign to a link i, we have that $c_{ij} = l$, and for any link $k \ne i$, $c_{kj} \ge (l-1)+1 = l$. Therefore, the coordination ratio is $\Omega(Ks)$. Since we can take $K = \Omega\left(1+\frac{\log m}{s\cdot\log(1+\frac{\log}{\ })}\right)$, it follows that $R = \Omega\left(s+\frac{\log m}{\log(1+\frac{\log}{\ })}\right)$.

We now handle the case when $s = \Omega(\log m)$. Consider the following problem instance: there are m tasks, where task j has weight 1 on link j, weight s on link $((j+1) \bmod m)$, and an infinite weight on any other link (assume the links are numbered $[0,\ldots,m-1]$).

Clearly, $OPT = 1$ by assigning task j to link j $(j = 0,\ldots,m-1)$. Consider the system of pure strategies S where task j is assigned to link $((j+1) \bmod m)$. Clearly, the system S is in Nash equilibrium. Moreover, the load on each link is s. Hence, the coordination ratio is $\Omega(s)$. □

We can modify the construction from Theorem 10 and obtain a tight lower bound for the mixed strategies case. The proof relies on the same arguments used in the proof of Theorem 7 and is omitted.

Theorem 11. *For mixed strategies,* $R = \Omega\left(s+\frac{s\cdot\log m}{\log(1+s\cdot\log\frac{\log}{\ })}\right)$.

References

1. R. Cole, Y. Dodis, and T. Roughgarden. How much can taxes help selfish routing? In *Proc. 4th Annual ACM Conference on Electronic Commerce*, pages 98–107, 2003.
2. R. Cole, Y. Dodis, and T. Roughgarden. Pricing network edges for heterogeneous selfish users. In *Proc. 35th ACM Symp. on Theory of Computing*, pages 521–530, 2003.
3. A. Czumaj, P. Krysta, and B. Vöcking. Selfish traffic allocation for server farms. In *Proc. 34th ACM Symp. on Theory of Computing*, pages 287–296, 2002.
4. A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, pages 413–420, 2002.
5. E. Koutsoupias and C.H. Papadimitriou. Worst-case equilibria. In *Proc. 16th Symp. on Theoretical Aspects of Comp. Science*, pages 404–413, 1999.

6. M. Mavronicolas and P. Spirakis. The price of selfish routing. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 510–519, 2001.
7. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, 1995.
8. C.H. Papadimitriou. Algorithms, games and the internet. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 749–753, 2001.
9. T. Roughgarden. Designing networks for selfish users is hard. In *Proc. 42nd IEEE Symp. on Found. of Comp. Science*, pages 472–481, 2001.
10. T. Roughgarden. How unfair is optimal routing? In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, pages 203–204, 2002.
11. T. Roughgarden. The price of anarchy is independent of the network topology. In *Proc. 34th ACM Symp. on Theory of Computing*, pages 428–437, 2002.
12. T. Roughgarden and É. Tardos. How bad is selfish routing. In *Proc. 41st IEEE Symp. on Found. of Comp. Science*, pages 93–102, 2000.

Load Balancing of Temporary Tasks in the ℓ_p Norm

Yossi Azar[1,⋆], Amir Epstein[1,⋆⋆], and Leah Epstein[2,⋆⋆⋆]

[1] School of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel
{azar,amirep}@tau.ac.il
[2] School of Computer Science, The Interdisciplinary Center, Herzliya, Israel
lea@idc.ac.il

Abstract. We consider the on-line load balancing problem where there are m identical machines (servers). Jobs arrive at arbitrary times, where each job has a weight and a duration. A job has to be assigned upon its arrival to exactly one of the machines. The duration of each job becomes known only upon its termination (this is called temporary tasks of unknown durations). Once a job has been assigned to a machine it cannot be reassigned to another machine. The goal is to minimize the maximum over time of the sum (over all machines) of the squares of the loads, instead of the traditional maximum load.

Minimizing the sum of the squares is equivalent to minimizing the load vector with respect to the ℓ_2 norm. We show that for the ℓ_2 norm greedy algorithm performs within at most 1.50 of the optimum. We show (an asymptotic) lower bound of 1.33 on the competitive ratio of the greedy algorithm. We also show a lower bound of 1.20 on the competitive ratio of any deterministic algorithm.

We extend our techniques and analyze the competitive ratio of greedy with respect to the ℓ_p norm. We show that the greedy algorithm performs within at most $2 - \Omega(1/p)$ of the optimum. We also show a lower bound of $2 - O(\ln p/p)$ on the competitive ratio of any on-line algorithm.

1 Introduction

We are given m parallel identical machines and a number of independent jobs (tasks) arriving at arbitrary times, where each job has a weight and a duration. A job should be assigned upon its arrival to exactly one of the machines based only on the previous jobs without any knowledge on the future jobs, thus increasing the **load** on this machine by its weight for the duration of the job. The duration of each job becomes known only upon its termination (this is called temporary tasks of unknown durations). The **load** of a machine is the sum of the weights of the jobs assigned to it. For any ℓ_p norm we define the **cost** of an assignment

⋆ Research supported in part by the Israel Science Foundation and by the IST program of the EU.
⋆⋆ Research supported in part by the Deutsch Institute.
⋆⋆⋆ Research supported in part by the Israel Science Foundation.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 53–66, 2004.
© Springer-Verlag Berlin Heidelberg 2004

for an input sequence of jobs as the maximum over time of the ℓ_p norm of the load vector. Specifically, the ℓ_∞ norm is the makespan (or maximum load) and the ℓ_2 norm is the Euclidean norm, which is equivalent to the sum of the squares of the load vector. The goal of an assignment algorithm is to assign all the jobs so as to minimize the cost.

Consider for example the case where the weight of a job corresponds to its machine disk access frequency. Each job may see a delay that is proportional to the load on the machine it is assigned to. Then the *average* delay is proportional to the sum of squares of the machines loads (namely the ℓ_2 norm of the corresponding machines load vector) whereas the *maximum* delay is proportional to the maximum load.

We measure the performance of an on-line algorithm by its **competitive ratio**. An on-line algorithm is c-competitive if for each input the cost of the assignment produced by the algorithm is at most c time larger than the cost of the optimal assignment.

We first summarize **our results**.

- For the ℓ_2 norm, we show that the greedy algorithm is at most 1.493-competitive (2.2293-competitive for the sum of the squares of loads) for any number of machines. In fact, for $m = 2$ greedy algorithm is optimal and its competitive ratio is 1.145 and for $m = 3$ we can improve the upper bound to 1.453.
- For the ℓ_2 norm, we show that there is no on-line algorithm that is 1.202-competitive (1.447-competitive for the sum of the squares of loads).
- For the ℓ_2 norm, we show a lower bound of $\frac{2}{\sqrt{3}} - O(\frac{1}{m})$ on the competitive ratio of any on-line algorithm for m machines and we show a lower bound of $\frac{2}{\sqrt{3}}$ for m divisible by 3.
- For the ℓ_2 norm, we show (an asymptotic) lower bound of 1.338 on the competitive ratio of the greedy algorithm.
- For the general ℓ_p norm (for any $p > 1$), we show that the greedy algorithm is at most $2 - \Omega(1/p)$-competitive for any number of machines.
- For the general ℓ_p norm (for any $p > 1$), we show (an asymptotic) lower bound of $2 - O(\ln p/p)$ on the competitive ratio of any on-line algorithm.
- For the general ℓ_p norm (for any $p > 1$), we show that for m=2 the greedy algorithm is an optimal on-line algorithm.

Temporary tasks, ℓ_∞ norm: For the problem of on-line load balancing of temporary tasks the upper bound is $2 - \frac{1}{m}$. This upper bound was proved for permanent tasks (tasks that never depart) by Graham [12], nevertheless Graham's analysis of the upper bound holds also for temporary tasks. The results in [4] show that his algorithm is optimal by constructing a lower bound of $2 - \frac{1}{m}$ on the competitive ratio of any deterministic on-line algorithm.

Permanent tasks, ℓ_∞ norm: The permanent tasks model is the model where tasks only arrive (on-line), but never depart. This is the classic ancient problem of scheduling jobs on identical machines minimizing the makespan (or maximum

load). Graham [12] showed that the greedy load balancing algorithm is $2-\frac{1}{m}$-competitive in this case. The greedy algorithm is an optimal on-line algorithm only for $m \leq 3$ [9].

Bartal et al. [6] were the first to show an algorithm whose competitive ratio is strictly below $c < 2$ (for all m). More precisely, their algorithm achieves a competitive ratio of $2 - \frac{1}{70}$. Later, the algorithm was generalized by Karger, Phillips and Torng [16] to yield an upper bound of 1.945. Subsequently, Albers [1] designed 1.923-competitive algorithm. Fleischer and Wahl [10] improved this result to a ratio of 1.9201.

Bartal et al. [7] showed a lower bound of 1.8370 for the problem. This result was improved by Albers [1] to a ratio of 1.852 and then by Gormley et al. [11] to a ratio of 1.853. The best lower bound currently known is due to Rudin [15], who showed a lower bound of 1.88.

Permanent tasks, ℓ_p norm: Chandra and Wong [8] were the first to consider the problem of minimizing the sum of squares of the machine loads. They showed that if the jobs arrive in non-increasing weights order then the greedy algorithm yields a schedule whose cost is within $\frac{25}{24}$ of the optimal cost. This result was slightly improved by Leung and Wei [17]. Chandra and Wong [8] also considered the general ℓ_p norm (for any $p > 1$) and showed that the greedy algorithm on the sorted items achieves a constant performance bound. The constant depends on p and grows to $\frac{3}{2}$ when p grows to ∞. The problem of on-line load balancing in the general ℓ_p norm (for any $p > 1$) for permanent tasks was considered in [3]. The results in [3] show that for the sum of the squares, the greedy algorithm performs within $\frac{4}{3}$ of the optimum, and no on-line algorithm achieves a better competitive ratio. For the sum of the squares [3] also provided on-line algorithm with competitive ratio $\frac{4}{3} - \delta$, for some fixed δ, for any sufficiently large number of machines. For the general ℓ_p norm the results show that the competitive ratio of greedy algorithm is $2 - \Theta((\ln p)/p)$.

Off-line results: Azar et al. [5] presented a polynomial time approximation scheme for the problem of off-line load balancing of temporary tasks (in the ℓ_∞ norm) in the case where the number of machines is fixed. For the case in which the number of machines is given as part of the input (i.e, not fixed) they showed that no polynomial algorithm can achieve a better approximation ratio than $\frac{3}{2}$ unless $P = NP$.

For the problem of off-line load balancing of permanent tasks (in the ℓ_∞ norm), there is a polynomial time approximation scheme for any fixed number of machines [13, 18] and also for arbitrary number of machines by Hochbaum and Shmoys [14].

Off-line scheduling and load balancing of permanent tasks with respect to the ℓ_p norm has been considered in [2]. The off-line minimization problem is known to be NP-hard in the strong sense. Alon et al. [2] provided a polynomial approximation scheme for scheduling jobs with respect to the ℓ_p norm for any $p > 1$. An example in which the optimal assignment for the sum of the squares is different than the optimal assignment in the ℓ_∞ norm is also given in [2].

2 Definitions and Preliminaries

In the **load balancing problem** we are given m identical machines (servers) and a finite sequence of events. We denote the input sequence by $\sigma = \sigma_1, \ldots, \sigma_r$. Each event σ_i is an arrival or departure of a job (task). We view σ as a sequence of times, the time σ_i is the moment after the $i^{\underline{th}}$ event happened. We denote the weight of a job j by w_j, its arrival time by a_j and its departure time (which is unknown until it departs) by d_j. An on-line algorithm has to assign a job upon its arrival without knowing the future jobs and the durations of jobs that have not departed yet. We compare the performance of on-line algorithms and an optimal off-line algorithm that knows the sequence of jobs and their durations in advance. Let $J_i = \{j \mid a_j \leq \sigma_i < d_j\}$ be the active jobs at time σ_i. A **schedule** S is an assignment which assigns each job j to a single machine k, $1 \leq k \leq m$. For every schedule S, the **load** of machine k at time σ_i, denoted $L_k^i(S)$, is the sum of weights of all jobs assigned to machine k in S, and active at this time. The **vector of loads at time** σ_i is $L^i(S) = (L_1^i(S), \ldots, L_m^i(S))$. Our cost measure is the ℓ_p norm. Hence the **cost** of a schedule S at time σ_i is defined as $\|L^i(S)\|_p = \left(\sum_{k=1}^{m} (L_k^i(S))^p\right)^{\frac{1}{p}}$. The **cost** of a schedule S is defined as $\|L(S)\|_p = \max_i \|L^i(S)\|_p$. We denote the load vector with the maximum cost, by $L(S) = (L_1(S), \ldots, L_m(S))$.

The **optimal cost**, denoted $OPT(S)$, is the minimal cost over all possible schedules for the given sequence of S.

We measure the performance of our algorithms by the **competitive ratio**. For a fixed $p > 1$, the **competitive ratio of a schedule** S is defined as $C(S) = \|L(S)\|_p / OPT(S)$. Let A be an on-line assignment algorithm. The **competitive ratio of** A **for a fixed number** $m \geq 1$ **of machines** is defined as

$$C_{A,m} = \sup\{C(S) \mid S \text{ is a schedule produced by } A \text{ on } m \text{ machines}\}.$$

The **competitive ratio of** A **for an arbitrary number of machines** is defined as $C_A = \sup\{C_{A,m} \mid m \geq 1\}$.

The previous definitions cover also the case where we measure the sum of squares of loads, since then the cost is $(\|L(S)\|_2)^2$. Consequently, the competitive ratios for the sum of the squares of loads are equal to $(C(S))^2$, $(C_{A,m})^2$ and $(C_A)^2$ w.r.t. the ℓ_2 norm.

Now we define the notion of a shape of a schedule, which is an abstraction of a schedule where for every machine, all jobs assigned to it except for one are replaced by very small jobs with the same total load. In general it may be impossible to produce such a schedule by the same algorithm as the original one. Nevertheless, the concept of a shape is very useful for proving upper bounds on the competitive ratio, since the optimal assignment may improve (by partitioning the jobs) while the cost of the assignment does not change. Hence a shape is a pessimistic estimate of a schedule. A shape characterizes each machine by two numbers, a_i is the total load of the small jobs, and u_i is (a lower bound on) the weight of one large job. We denote a **shape** by a pair $R = (a, u)$, where a and u are vectors of m nonnegative reals. The **vector of loads** of a

shape is defined as $L(R) = a + u$. The **competitive ratio** of a shape R is $C(R) = \|L(R)\|_p / OPT(R)$.

It is possible to compute the optimal cost of the shape $R = (a, u)$ explicitly. It is the cost of a schedule in which some big jobs are scheduled each on a separate machine and the rest of the jobs are balanced evenly on the rest of the machines. Let the machines be ordered so that u_i are nondecreasing. For $1 \leq l \leq m$ let $h_l = (\sum_{i=1}^{m} a_i + \sum_{i=1}^{l} u_i)/l$. Let k be the largest l such that $h_l \geq u_l$ (k is always defined, since $h_1 \geq u_1$). We define the **height** of the shape to be $h(R) = h_k$.

It is easy to see that a good candidate for an optimal schedule for the shape R is to put on each machine one job of size exactly u_i and partition a_i into a few jobs so that they can be balanced exactly on the k machines; then the load vector is $(h_k, \ldots, h_k, u_{k+1}, \ldots, u_m)$. See the Figures 1 and 2 for examples where $a_i = 1$ for all i.

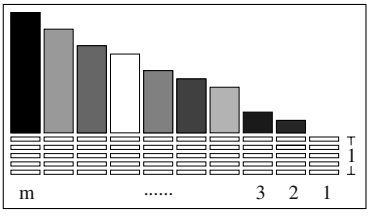


Fig. 1. A shape R.

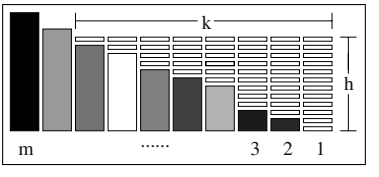


Fig. 2. Optimal assignment of R.

Now we extend the notion of shape and define continuous shapes, which are defined similarly to shapes. It is an extension, which treats the machines as points in the interval $[0, m]$. The load of machines is a function defined on that interval and is the sum of two functions. One function is the total load of very small jobs on each machine. The other function is the load of one big job on each machine. Formally a **continuous shape** is a pair $R = (a, u)$, where a and u are functions (not necessarily continuous), defined in the interval $[0, m]$. The **function of loads** of a shape is defined as $L(R) = a + u$. Where $a(t)$ represents the total load of very small jobs at point t in the interval $[0, m]$ and $u(t)$ represents the load of one big job at point t in the interval $[0, m]$. From the convexity of the function x^p it follows that the **optimal cost of a continuous shape** R is obtained by assigning some big jobs each on a separate machine and the remaining jobs are balanced evenly on the rest of the machines. Formally w.l.o.g let $u(t)$ be a non-decreasing function. There exists t' , s.t $u'(t) = u(t)$ and

$$a'(t) = \begin{cases} 0 & t \leq t' \\ \frac{1}{m-t'}(\int_0^m a(t)dt + \int_{t'}^m u(t)dt) - u(t) & t' < t \end{cases}$$

and for t_1, t_2, s.t $t_1 \leq t' < t_2$, it holds that $u'(t_1) \geq a'(t_2) + u'(t_2)$ and $R' = (a', u')$ gives the optimal load $L(R')$ for the shape R.

Transition from a shape to a continuous shape is defined as follows. Let $R = (a, u)$ be a shape, then its continuous shape $R' = (a', u')$ is

$$a'(t) = \{ a_i \quad i - 1 < t \leq i$$
$$u'(t) = \{ u_i \quad i - 1 < t \leq i \ .$$

Note that when moving from a schedule to a shape and from a shape to a continuous shape the cost of the assignment does not change while the cost of the optimal assignment can only decrease.

3 The Greedy Algorithm

In this section we analyze the competitive ratio of the greedy algorithm defined below.

> **Algorithm Greedy:** Upon arrival of a job j assign it to the machine with the current minimum load (ties are broken arbitrarily).

Note that a job departs from a machine it was assigned to.

To obtain a constant upper bound for the performance ratio of Greedy, we show that each schedule can be replaced by a very special continuous shape so that the competitive ratio does not decrease. Computing the competitive ratio is then shown to be equivalent to computing the maximum of a certain function with equality constraints over the reals. A shape $R = (h, x)$ is called **partial flat** if there exists an integer $1 \leq k \leq m$, and a real $c > 0$ such that the following conditions hold:

$$h_i = c \qquad \text{for } i = 1, \ldots, k$$
$$x_i \geq 0 \qquad \text{for } i = 1, \ldots, k$$
$$h_i = x_i = 0 \text{ for } i = k+1, \ldots, m.$$

When $k = m$ the shape is called **flat**. A shape $R = (h, x)$ is called **separate** if there exists an integer $1 \leq k \leq m$ and a real $c > 0$, such that the following conditions hold:

$$h_i = 0 \text{ for } i = 1, \ldots, k$$
$$x_i \geq c \text{ for } i = 1, \ldots, k$$
$$h_i = c \text{ for } i = k+1, \ldots, m$$
$$x_i = 0 \text{ for } i = k+1, \ldots, m.$$

Let S be a schedule obtained by Greedy and let $L(S)$ be the load when Greedy reaches the maximum cost. Let h be the load vector of all jobs except the last job assigned to each machine, we treat these jobs as very small jobs and call them sand jobs. Let x be the weight vector of the last job assigned to each machine. The shape $R = (h, x)$ is a shape of the schedule S, we call it the **Greedy** shape. Recall that when moving from a schedule to a shape and from a shape to a continuous shape the cost of the assignment does not change while the cost of the optimal assignment can only decrease.

Lemma 1. *Let $R = (h, x)$ be the Greedy shape of a Greedy schedule S, normalized such that $(OPT(S))^p = m$. Then there exists a partial flat shape $R' = (h', x')$ such that $\|L(R)\|_p \leq \|L(R')\|_p$ and $OPT(R) \geq OPT(R')$ with the non-zero components of h' equal to 1, and the off-line shape of the shape R' is a separate shape.*

Proof. It is easy to see that $h_i \leq 1$ (otherwise when the last job was assigned to machine i before Greedy reached the maximum cost, $(OPT(S))^p > m$, which is a contradiction).

W.l.o.g we assume that the machines are ordered so that $h_i + x_i$ are non increasing. We perform the following transformation. Note that in each step of the transformation the cost of greedy can only increase and the cost of the off-line algorithm can only decrease. This is in particular due to the convexity of the function x^p. Figure 3 shows the obtained shape at the end of each transformation step.

1. In this step we move to a continuous shape $R = (h(t), x(t))$, where $h(t)$ and $x(t)$ are functions in the interval $[0, m]$. We treat each point t in that interval as a machine and the value $h(t) + x(t)$ as its load. Now we transform regular jobs (jobs or part of jobs) that are placed below height 1 to sand jobs. Next we push the sand jobs left such that the sand jobs height will be equal to 1 from point 0 to point V_0, where V_0 is the total volume of the sand jobs. Formally, let $R = (h(t), x(t))$ be the current shape and let t_0 be maximal, such that $h(t) + x(t) \geq 1$ then the new shape $R' = (h'(t), x'(t))$ is obtained as follows. Denote
$$V_0 = t_0 + \int_{t_0}^{m} (h(t) + x(t))dt$$
then
$$h'(t) = \begin{cases} 1 & t \leq V_0 \\ 0 & V_0 < t \end{cases}$$
$$x'(t) = \begin{cases} h(t) + x(t) - 1 & t \leq t_0 \\ 0 & t_0 < t \end{cases}$$
2. Jobs that in the off-line algorithm are scheduled on machines with sand jobs are pushed left to have the same height as the load of those machines in the off-line algorithm (which are balanced). Formally, let $R = (h(t), x(t))$ be the current shape then the new shape $R' = (h'(t), x'(t))$ is obtained as follows. Let t_1 be a minimal point such that machine t_1 has sand jobs in the off-line algorithm, let w be its total load in the off-line algorithm and let V_1 be the volume of the regular jobs on machines $[t_1, m]$. Denote
$$V_1 = \int_{t_1}^{m} x(t)dt$$
then
$$h'(t) = h(t)$$

$$x'(t) = \begin{cases} x(t) & t \le t_1 \\ w & t_1 < t \le t_1 + \frac{V_1}{w} \\ 0 & \text{otherwise} \end{cases}$$

3. Sand jobs on machines with no regular jobs are transformed continuously to regular jobs of height equal to w (as defined in the previous step). We put these jobs on the leftmost machines possible that have only sand jobs. We perform this process from right to left. This process continues until all machines in the greedy shape have sand jobs and a regular job. Formally, let $R = (h(t), x(t))$ be the current shape then the new shape $R' = (h'(t), x'(t))$ is obtained as follows. Let t_1 be maximal such that machine t_1 has sand jobs and a regular job. Let t_2 be maximal such that machine t_2 has jobs (any jobs). Let $s = \frac{t_1 \cdot w + t_2}{w+1}$, then

$$h'(t) = \begin{cases} 1 & t \le s \\ 0 & \text{otherwise} \end{cases}$$

$$x'(t) = \begin{cases} x(t) & t \le t_1 \\ w & t_1 < t \le s \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that in each of the transformation steps the cost of greedy can only increase and the cost of the off-line algorithm can only decrease due to the convexity of the function x^p. We denote the shape obtained by the transformation by $R' = (h', x')$. This shape has jobs on machines $[0, s]$ for some real number $0 < s < m$. Each machine $t \in [0, s]$ has sand jobs of height 1 and a regular job of height $x'(t) > 0$, other machines have no jobs assigned to them, hence this is a partial flat shape. The off-line shape has jobs of height $x'(t)$ on machines $t \in [0, s]$ and sand jobs of total volume s evenly assigned to machines $(s, m]$. In addition $\min_{t \in [0,s]} x'(t) \ge w = s/(m - s)$, hence the off-line shape is a separate shape. This completes the proof.

Lemma 2. *Let $R = (h, x)$ be a partial flat shape such that $h(t) = 1$ for $0 \le t \le s$. Assume that the off-line shape of R is a separate shape. Then there is a shape R'=(h,x'), such that for $0 \le t \le s$, $x'(t) = y$ for some value $y > 0$ and otherwise $x'(t) = 0$ and it holds that $\|L(R)\|_p \le \|L(R')\|_p$ and $OPT(R) = OPT(R')$.*

Proof. Let $\delta \cdot m = s$. Define

$$y = (\frac{1}{\delta \cdot m} \int_0^{\delta \cdot m} x^p(t)dt)^{\frac{1}{}}.$$

Clearly $OPT(R) = OPT(R')$. Now

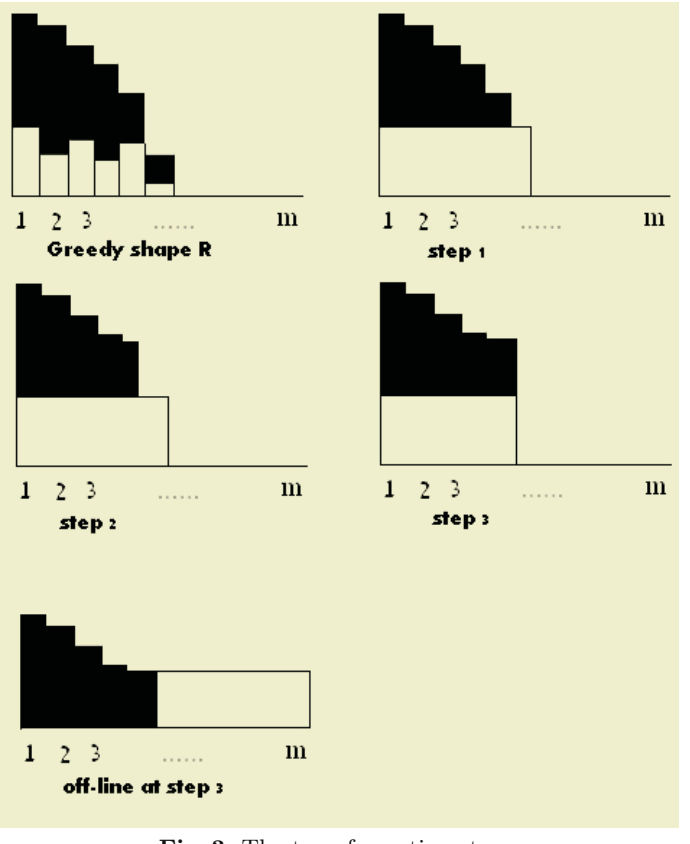


Fig. 3. The transformation steps.

$$\begin{aligned}
\|L(R)\|_p &= \left(\int_0^{\delta \cdot m} (1 + x(t))^p dt\right)^{\frac{1}{p}} \\
&\leq \left(\int_0^{\delta \cdot m} 1 dt\right)^{\frac{1}{p}} + \left(\int_0^{\delta \cdot m} x^p(t) dt\right)^{\frac{1}{p}} \\
&= (\delta \cdot m)^{\frac{1}{p}} + (\delta \cdot m)^{\frac{1}{p}} \cdot y \\
&= (\delta \cdot m)^{\frac{1}{p}} \cdot (1 + y) \\
&= \|L(R')\|_p
\end{aligned}$$

where the inequality follows from the triangle inequality for the ℓ_p norm. This completes the proof.

Define the function $f(\delta, x)$ with the constraint $g(\delta, x)$.

$$f(\delta, x) = \delta \cdot (1 + x)^p, \tag{1}$$

$$g(\delta, x) = \delta \cdot x^p + \frac{\delta^p}{(1 - \delta)^{p-1}} = 1. \tag{2}$$

Theorem 1. *The competitive ratio of Greedy algorithm satisfies*

$$C_{Greedy} \leq (f_{max})^{\frac{1}{}}$$

where f_{max} is the maximum of f with constraint (2), in the domain $0 \leq x, 0 \leq \delta \leq \frac{1}{2}$.

Proof. Let R_0 be the Greedy shape of a schedule S obtained by Greedy. For simplicity we transform to a new shape R_1 by normalizing all job weights such that

$$(OPT(R_1))^p = m. \tag{3}$$

If all the h components of R_1 are equal to zero then the Greedy schedule is optimal. Otherwise by applying Lemma 1 and Lemma 2 we obtain from R_1 a partial flat shape $R_2 = (h, x)$, in which all the non zero components of x are the same and its off-line shape is a separate shape such that $\|L(R_1)\|_p \leq \|L(R_2)\|_p$ and $OPT(R_1) \geq OPT(R_2)$. We have

$$(\|L(R_2)\|_p)^p = \delta \cdot m \cdot (1+x)^p, \tag{4}$$

$$\begin{aligned}(OPT(R_2))^p &= \int_0^{\delta \cdot m} x^p(t)dt + (1-\delta)m(\frac{\delta \cdot m}{(1-\delta)\cdot m})^p \\ &= \delta \cdot m \cdot x^p + \frac{\delta^p \cdot m}{(1-\delta)^{p-1}}, \end{aligned} \tag{5}$$

$$x \geq \frac{\delta \cdot m}{(1-\delta)\cdot m} = \frac{\delta}{1-\delta}. \tag{6}$$

The last inequality restricts the weight of a regular job to be greater then the total weight of sand jobs on machines with sand jobs in the off-line shape. For simplicity we divide equalities (4) and (5) by m, this does not change the ratio between the cost and the off-line cost of shape R_2. Which gives the following

$$f(\delta, x) = \delta \cdot (1+x)^p, \tag{7}$$

$$1 \geq \delta \cdot x^p + \frac{\delta^p}{(1-\delta)^{p-1}}, \tag{8}$$

$$x \geq \frac{\delta}{1-\delta}. \tag{9}$$

The left side of the equality is $f(\delta, x)$ by definition. The first inequality results from (5), since $(OPT(R_2))^p \leq (OPT(R_1))^p \leq m$ and the division by m.

Substituting (9) in (8) gives

$$1 \geq \frac{\delta^{p+1}}{(1-\delta)^p} + \frac{\delta^p}{(1-\delta)^{p-1}} = \frac{\delta^p}{(1-\delta)^p}$$

which yields $\delta \leq \frac{1}{2}$. We obtain the following relation for f

$$f(\delta, x) = \frac{\frac{1}{m}(\|L(R_2)\|_p)^p}{\frac{1}{m}\cdot m} \geq \frac{(\|L(R_1)\|_p)^p}{(OPT(R_1))^p} \geq \frac{(\|L(S)\|_p)^p}{(OPT(S))^p}$$

where the first inequality follows from (3) and the fact that $\|L(R_1)\|_p \leq \|L(R_2)\|_p$. Hence to bound the competitive ratio of Greedy we need to solve the following maximum problem in the domain $0 \leq \delta \leq \frac{1}{2}$ and $0 \leq x$. We need to find the maximum of $f(\delta, x)$ under the constraint (8). It is easy to see that the maximum of f is obtained when (8) is an equality. Hence we obtain the following maximum problem for f with constraint g (1), (2) in the domain $0 \leq x, 0 \leq \delta \leq \frac{1}{2}$. This completes the proof.

The following theorem results from Theorem 1.

Theorem 2. *For ℓ_2 norm the competitive ratio of Greedy algorithm is $C_{Greedy} \leq$ 1.493.*

Proof. Omitted.

The upper bound can be improved for 3 machines.

Theorem 3. *For ℓ_2 norm and $m = 3$ the competitive ratio of Greedy algorithm, $C_{Greedy,3} \leq 1.453$.*

Proof. Omitted.

Now we turn to the case of the general ℓ_p norm.

Theorem 4. *For any $p > 1$ it holds $C_{Greedy} \leq 2 - \Omega\left(\frac{1}{p}\right)$.*

Proof. Omitted.

Theorem 5. *For $m = 2$ the greedy algorithm is optimal and its competitive ratio is*

$$C_{Greedy,2} = \sup_{x \geq 0} \left(\frac{1 + (1+x)^p}{2^p + x^p} \right)^{\frac{1}{p}}, \tag{10}$$

and the supremum is achieved as a maximum at the unique solution $x \in (0, \infty)$ of the equation

$$x^{p-1}(1 + (1+x)^{1-p}) = 2^p.$$

Proof. Omitted.

From the above theorem, which claims that for $m = 2$ Greedy is optimal and its competitive ratio is the same as in the permanent tasks case, we obtain according to [3] that for the ℓ_2 norm and for two machines Greedy is optimal and its competitive ratio is $\sqrt{(\sqrt{5}+3)/4} \approx 1.145$.

4 Lower Bounds

4.1 Lower Bounds for ℓ_2 Norm

In this section we give lower bounds for the ℓ_2 norm. We prove a lower bound for any algorithm (the proof is for $m = 3$). Then we prove a weaker lower bound for any $m \geq 3$. Finally we prove a lower bound for Greedy for a large number of machines ($m \to \infty$).

Theorem 6. *For any on-line assignment algorithm A, $C_A \geq C_{A,3} \geq 1.202$.*

Proof. Consider the following sequence for three machines. First three unit jobs and one job of weight $x \geq 1$ arrive. Then two unit jobs depart. At last one job of weight 2 arrive. Consider the first three unit jobs. If algorithm A assigns two or more jobs to the same machine, it does not get any other job. Its cost is at least 5, the optimal cost is 3, and we are done. Otherwise, algorithm A assigns one unit job to every machine (the off-line algorithm assigns two unit jobs to machine 1 and one unit job to machine 2). Now the next job of weight x arrives. Algorithm A assigns it to one of the machines say 1 (the off-line algorithm assigns it to machine 3). Then two unit jobs depart, which are the jobs on machines 2,3 (the jobs on machine 1 in the off-line algorithm). At last a job of weight 2 arrive. The best algorithm A can do is to assign it to one of the empty machines 2 or 3 (the off-line algorithm assigns it to machine 1). Its cost is at least $\sqrt{(1+x)^2+2^2}$, whereas the optimum cost is $\sqrt{2^2+1+x^2}$. The maximal ratio ≈ 1.202 is achieved for $x = \sqrt{5}$.

Theorem 7. *For any number of machines $m \geq 3$ and any on-line assignment algorithm A, $C_{A,m} \geq 2/\sqrt{3} - O(\frac{1}{m})$. For m divisible by 3, $C_{A,m} \geq 2/\sqrt{3}$.*

Proof. Let $m = 3k$. We consider the following sequence. First $4k$ unit jobs arrive. Then $2k$ unit jobs depart. Finally k jobs of weight 2 arrive. Consider the arrival of the first $4k$ unit jobs. Algorithm A assigns these jobs. W.l.o.g we assume that machines $1, \ldots, m$ are sorted in nondecreasing order of load (the off-line algorithm assigns two jobs on each machine $1, \ldots, k$ and one job on each machine $k+1, \ldots, 3k$). Then $2k$ jobs depart. There exists a minimal $t \geq 2k$, such that machines $1, \ldots, t$ are assigned at least $2k$ jobs. Then $2k$ jobs from machines $1, \ldots, t$ depart as follows, all jobs from machines $1, \ldots, t-1$ and some jobs from machine t (in the off-line algorithm the jobs on machines $1, \ldots, k$ depart). At the end of this step machines $1, \ldots, 2k$ are empty. Next k jobs of weight 2 arrive. The best algorithm A can do is to assign each job to an empty machine (In the off-line these jobs are assigned to machines $1, \ldots, k$). Finally there are jobs of total weight $4k$ assigned to no more than $2k$ machines. Due to the convexity of the function x^p, the minimum cost is obtained when all machines have the same load, therefore its cost is at least $\sqrt{2k \cdot (2)^2}$. The optimum cost is $\sqrt{k \cdot 2^2 + 2k \cdot 1^2}$, which yields a ratio of $2/\sqrt{3}$. For m not divisible by 3 a similar proof gives a ratio of $2/\sqrt{3} - O(\frac{1}{m})$.

Theorem 8. *For the greedy algorithm, $C_{Greedy} \geq 1.338$.*

Proof. First we prove a weaker lower bound of 1.314 for the greedy algorithm, by solving an ordinary differential equation analytically. Then by a similar proof we obtain a more complex ordinary differential equation, which has no simple solution. Hence we use a computer program to compute the competitive ratio in this case, which gives a lower bound of 1.338.

We start with the first proof. We see the m machines as m points in the interval $(0, 1]$, machine i as the point $\frac{i}{m} \in (0, 1]$, and the load of the machines as a function $f(t)$, $f(t) = l_i$ for $\frac{(i-1)}{m} < t \leq \frac{i}{m}$, where l_i is the load of machine i. For each machine i the total load is the value of f in the interval $(\frac{i-1}{m}, \frac{i}{m}]$ and the total load of all machines is the total volume of f in the interval $(0, 1]$ multiplied by m. Let $f(k/m)$ be the load of machine k at the end of step k and let $F(k/m)$ be the volume of the jobs assigned to machines $k, \ldots, m$ at the beginning of step k in the following process. For convenience we number the steps from m to 1 in decreasing order. In this process we keep the volume of jobs fixed and equal to 1 at the end of each step. We start with the arrival of infinitesimally small jobs of total volume 1, we call jobs of this type sand jobs. Both the off-line and greedy algorithms assign these jobs evenly on all the machines (total height 1 on each machine). At step k a job of height x ($x \geq 1$) arrives. Greedy assigns this job to the machine with minimum load w.l.o.g to machine k which is the one with the largest index among all machines with the minimum load $1, \ldots, k$ (otherwise we swap indices) and the off-line algorithm performs the same assignment. Then the sand jobs on machines $1, \ldots, k-1$ depart in greedy. In the off-line algorithm the departing jobs are composed of all the sand jobs of machine k and equal amounts of sand jobs from machines $1, \ldots, k-1$ with the appropriate volume. Next sand jobs arrive with total volume $1 - F(k/m) - \frac{x}{m}$ (=1-total volume of machines $k, \ldots, m$), thus keeping the total volume equal to 1. Greedy and the off-line algorithms assign the sand jobs to machines $1, \ldots, k-1$ evenly, such that these machines will have the same load. At the end of step k

$$f(k/m) = \frac{1 - F(k/m)}{k/m} + x.$$

The computation of the lower bound for Greedy according to the above scenario, which is technical, is omitted.

4.2 Lower Bound for General $p > 1$

In this section we construct a lower bound for general $p > 1$.

Theorem 9. *For any $p > 1$ and any on-line algorithm A it holds that $C_A \geq 2 - O\left(\frac{\ln p}{p}\right)$.*

Proof. Omitted.

Acknowledgment

We would like to thank Adi Avidor for letting us use Figures 1 and 2.

References

1. S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29:459–473, 1999.
2. N. Alon, Y. Azar, G. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.
3. A. Avidor, Y. Azar, and J. Sgall. Ancient and new algorithms for load balancing in the ℓ_p norm. *Algorithmica*, 29:422–441, 2001. Also in *Proc. 9th ACM-SIAM SODA*, 1998, pp. 426-435.
4. Y. Azar and L. Epstein. On-line load balancing of temporary tasks on identical machines. In *5th Israeli Symp. on Theory of Computing and Systems*, pages 119–125, 1997.
5. Y. Azar, O. Regev, J. Sgall, and G. Woeginger. Off-line temporary tasks assignment. *Theoretical Computer Science*, 287:419–428, 2002.
6. Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51(3):359–366, 1995.
7. Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Information Processing Letters*, 50:113–116, 1994.
8. A.K. Chandra and C.K. Wong. Worst-case analysis of a placement algorithm related to storage allocation. *SIAM Journal on Computing*, 4(3):249–263, 1975.
9. U. Faigle, W. Kern, and G. Turan. On the performance of online algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.
10. R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling*, 3(5):343–353, 2000.
11. T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 564–565, 2000.
12. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
13. R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:416–429, 1969.
14. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. Assoc. Comput. Mach.*, 34(1):144–162, January 1987.
15. J.F. Rudin III. *Improved bounds for the on-line scheduling problem.* PhD thesis, The University of Texas at Dallas, May 2001.
16. D. Karger, S. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, 1996.
17. J.Y.T. Leung and W.D. Wei. Tighter bounds on a heuristic for a partition problem. *Information Processing Letters*, 56:51–57, 1995.
18. S. Sahni. Algorithms for scheduling independent tasks. *Journal of the Association for Computing Machinery*, 23:116–127, 1976.

Simple On-Line Algorithms for Call Control in Cellular Networks*

Ioannis Caragiannis, Christos Kaklamanis, and Evi Papaioannou

Computer Technology Institute and
Dept. of Computer Engineering and Informatics
University of Patras, 26500 Rio, Greece

Abstract. We address an important communication issue in wireless cellular networks that utilize Frequency Division Multiplexing (FDM) technology. In such networks, many users within the same geographical region (cell) can communicate simultaneously with other users of the network using distinct frequencies. The spectrum of the available frequencies is limited; thus, efficient solutions to the call control problem are essential. The objective of the call control problem is, given a spectrum of available frequencies and users that wish to communicate, to maximize the number of users that communicate without signal interference. We consider cellular networks of reuse distance $k \geq 2$ and we study the on-line version of the problem using competitive analysis.
In cellular networks of reuse distance 2, the previously best known algorithm that beats the lower bound of 3 on the competitiveness of deterministic algorithms works on networks with one frequency, achieves a competitive ratio against oblivious adversaries which is between 2.469 and 2.651, and uses a number of random bits at least proportional to the size of the network. We significantly improve this result by presenting a series of simple randomized algorithms that have competitive ratios smaller than 3, work on networks with arbitrarily many frequencies, and use only a constant number of random bits or a comparable weak random source. The best competitiveness upper bound we obtain is 7/3.
In cellular networks of reuse distance $k > 2$, we present simple randomized on-line call control algorithms with competitive ratios which significantly beat the lower bounds on the competitiveness of deterministic ones and use only $O(\log k)$ random bits. Furthermore, we show a new lower bound on the competitiveness of on-line call control algorithms in cellular networks of reuse distance $k \geq 5$.

1 Introduction

In this paper we study frequency spectrum management issues in wireless networks. Due to the rapid growth of wireless communications and the limitations and cost of the frequency spectrum, such issues are very important nowadays.

* This work was partially funded by the European Union under IST FET Project ALCOM–FT, IST FET Project CRESCCO and RTN Project ARACNE.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 67–80, 2004.
© Springer-Verlag Berlin Heidelberg 2004

We consider wireless networks in which base stations are used to build the required infrastructure. In such systems, the architectural approach used is the following. A geographical area in which communication takes place is divided into regions. Each region is the calling area of a base station. Base stations are connected via a high speed network.When a user A wishes to communicate with some other user B, a path must be established between the base stations of the regions in which the users A and B are located. Then communication is performed in three steps: (a) wireless communication between A and its base station, (b) communication between the base stations, and (c) wireless communication between B and its base station. Thus, the transmission of a message from A to B first takes place between A and its base station, the base station of A sends the message to the base station of B which will transmit it to B. At least one base station is involved in the communication even if both users are located in the same region or only one of the two users is part of the wireless network (and the other uses for example the PSTN).Improving the access of users to base stations is the aim of this work.

The network topology usually adopted [6, 7] is the one shown in the left part of Figure 1. All regions are regular hexagons (cells) of the same size. This shape results from the uniform distribution of identical base stations within the network, as well as from the fact that the calling area of a base station is a circle which, for simplicity reasons, is idealized as a regular hexagon. Due to the shape of the regions, we call these networks cellular wireless networks.

Many users of the same region can communicate simultaneously with their base station. This can be achieved via frequency division multiplexing (FDM). The base station is responsible for allocating distinct frequencies from the available spectrum to users so that signal interference is avoided. Since the spectrum of available frequencies is limited, important engineering problems related to the efficient reuse of frequencies arise. Signal interference usually manifests itself when the same frequency is assigned to users located in the same or adjacent cells. Alternatively, in this case, we may say that the cellular network has *reuse distance* 2. By generalizing this parameter, we obtain cellular networks of reuse distance k in which signal interference between users that have been assigned the same frequency is avoided only if the users are located in cells with distance at least k.

Signal interference in cellular networks can be represented by an *interference graph* G. Vertices of the graph correspond to cells and an edge (u, v) in the graph indicates that the assignment of the same frequency to two users lying at the cells corresponding to nodes u and v will cause signal interference. The interference graph of a cellular network of reuse distance 2 is depicted in the right part of Figure 1. If the assumption of uniform distribution of identical base stations does not hold, arbitrary interference graphs can be used to model the underlying network.

In this paper we study the *call control* problem which is defined as follows: Given users that wish to communicate with their base station, the *call control* problem on a network that supports a spectrum of w available frequencies is to

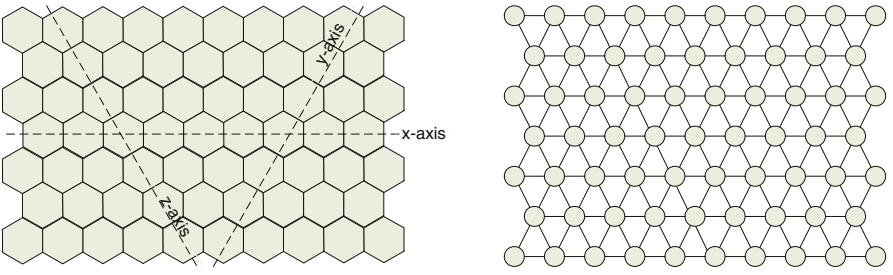


Fig. 1. A cellular network and the interference graph if the reuse distance is 2.

assign frequencies to users so that at most w frequencies are used in total, signal interference is avoided, and the number of users served is maximized.

We assume that calls corresponding to users that wish to communicate appear in the cells of the network in an on–line manner. When a call arrives, a call control algorithm decides either to accept the call (assigning a frequency to it), or to reject it. Once a call is accepted, it cannot be rejected (preempted). Furthermore, the frequency assigned to the call cannot be changed in the future. We assume that all calls have infinite duration; this assumption is equivalent to considering calls of the same duration.

Competitive analysis [11] has been used for evaluating the performance of on–line algorithms for various problems. In our setting, given a sequence of calls, the performance of an on–line algorithm A is compared to the performance of the optimal algorithm OPT.

Let $B(\sigma)$ be the benefit of the on–line algorithm A on the sequence of calls σ, i.e. the set of calls of σ accepted by A and $O(\sigma)$ the benefit of the optimal algorithm. If A is a deterministic algorithm, we define its competitive ratio (or competitiveness) as $\max_\sigma \frac{|O(\sigma)|}{|B(\sigma)|}$, where the maximum is taken over all possible sequences of calls. If A is a randomized algorithm, we define its competitive ratio as $\max_\sigma \frac{|O(\sigma)|}{\mathcal{E}[|B(\sigma)|]}$, where $\mathcal{E}[|B(\sigma)|]$ is the expectation of the number of calls accepted by A, and the maximum is taken over all possible sequences of calls.

Usually, we compare the performance of deterministic algorithms against *off–line adversaries*, i.e. adversaries that have knowledge of the behaviour of the deterministic algorithm in advance. In the case of randomized algorithms, we consider *oblivious adversaries* whose knowledge is limited to the probability distribution of the random choices of the randomized algorithm.

The static version of the call control problem is very similar to the famous maximum independent set problem. The on–line version of the problem is studied in [1–4, 8, 10]. [1], [2], and [8] study the call control problem in the context of optical networks. Pantziou et al. [10] present upper bounds for networks with planar and arbitrary interference graphs. Usually, competitive analysis of call control focuses on networks supporting one frequency. Awerbuch et al. [1] present a simple way to transform algorithms designed for one frequency to algorithms for arbitrarily many frequencies with a small sacrifice in competitiveness (see

also [5, 12]). Lower bounds for call control in arbitrary networks are presented in [3].

The greedy algorithm is probably the simplest on-line algorithm. When a call arrives, the greedy algorithm seeks for the first available frequency. If such a frequency exists, the algorithm accepts the call assigning this frequency to it, otherwise, the call is rejected. In general, Pantziou et al. [10] show that this algorithm has competitive ratio equal to the degree of the interference graph and no better in general. The greedy algorithm is optimal within the class of deterministic on-line call control algorithms.

Simple randomized algorithms can be defined using the "classify and randomly select" paradigm [1, 2, 10]. Such algorithms use a coloring of the underlying interference graph, randomly select a color out of the colors used, and execute the greedy algorithm in the cells colored with the selected color, ignoring (i.e., rejecting) calls in all other cells. The competitive ratio achieved in this way, against oblivious adversaries, is equal to the number of colors used in the coloring of the interference graph.

In cellular networks of reuse distance 2, the greedy algorithm is 3-competitive against off-line adversaries, in the case of one frequency. Slightly worse competitiveness bounds can be proved in the case of arbitrarily many frequencies using the techniques of [1, 5, 12]. In [4], using similar arguments with those of [10], it was observed that no deterministic on-line call control algorithm in cellular networks of reuse distance 2 can be better than 3-competitive against off-line adversaries. Applying the "classify and randomly select" paradigm using a 3-coloring of the interference graph, we obtain a 3-competitive randomized algorithm even in the case of arbitrarily many frequencies. Observe that this algorithm uses a very weak random source which equiprobably selects one out of three distinct objects.

The authors in [4] describe algorithm p–RANDOM, an intuitive on–line randomized call control algorithm for networks that support one frequency. They present upper and lower bounds on the competitive ratio of the algorithm as functions of parameter p and, by optimizing these functions, they prove that, for some value of p, the competitive ratio of algorithm p-RANDOM against oblivious adversaries is between 2.469 an 2.651. The analysis of algorithm p-RANDOM in [4] applies only to cellular networks with one frequency but it indicates that randomization helps to beat the deterministic upper bounds. However, the number of random bits used by the algorithm may be proportional to the size of the network. The best known lower bound on the competitive ratio of any randomized call control algorithm in cellular networks of reuse distance 2 is 1.857 [4].

Results on on-line call control in cellular networks of reuse distance $k > 2$ are only implicit in previous work. The greedy algorithm has competitive ratio 4 and 5 in cellular networks of reuse distance $k \in \{3, 4, 5\}$ and $k \geq 6$, respectively, which support one frequency. This is due to the fact that the acceptance of a non-optimal call may cause the rejection of at most 4 and 5 optimal calls, respectively. These competitive ratios are the best possible that can be achieved by deterministic algorithms. Using the techniques of [1, 5, 12], it can be shown that,

in the case of arbitrarily many frequencies, the greedy algorithm has competitive ratio at most 4.521 and at most 5.517 in cellular networks of reuse distance $k \in \{3, 4, 5\}$ and $k \geq 6$, respectively.

Furthermore, applying the "classify and randomly select" paradigm using an efficient coloring of the interference graph of cellular networks of distance reuse $k > 2$ would give randomized on-line algorithms with competitive ratio $\Omega(k^2)$. Even in the case of $k = 3$, the competitive ratio we obtain in this way is 7.

In this paper, we improve previous results on the competitiveness of on-line call control algorithms in cellular networks. We present algorithms based on the "classify and randomly select" paradigm which use new colorings of the interference graph. These algorithms use a small number of random bits, and have small competitive ratios against oblivious adversaries even in the case of arbitrarily many frequencies. In particular, in cellular networks of reuse distance 2, we significantly improve the best known competitiveness bounds achieved by algorithm p-RANDOM by presenting a series of simple randomized algorithms that have smaller competitive ratios, work on networks with arbitrarily many frequencies, and use only a constant number of random bits or a comparable weak random source. The best competitiveness upper bound we obtain is 7/3. In cellular networks of reuse distance $k > 2$, we present simple randomized on-line call control algorithms with competitive ratios which significantly beat the lower bounds on the competitiveness of deterministic algorithms and use only $O(\log k)$ random bits. For any $k > 2$, the competitive ratio we achieve is strictly smaller than 4. Furthermore, we show a new lower bound of 25/12 on the competitiveness, against oblivious adversaries, of on-line call control algorithms in cellular networks of reuse distance $k \geq 5$.

The rest of the paper is structured as follows. We present a simple 8/3-competitive randomized algorithm in Section 2. Simpler algorithms with better competitive ratios (including the algorithm with competitive ratio 7/3) are presented in Section 3. The upper bounds for cellular networks of reuse distance $k > 2$ are presented in Section 4. A Our new lower bound for cellular networks of reuse distance $k \geq 5$ can be found in Section 5.

2 A Simple 8/3-Competitive Algorithm

In this section we present algorithm CRS-A, a simple randomized on-line algorithm for call control in cellular networks of reuse distance 2. The algorithm works in networks with one frequency and achieves a competitive ratio against oblivious adversaries which is similar (but slightly inferior) to that which has been proved for algorithm p-RANDOM.

Algorithm CRS-A uses a coloring of the cells with four colors 0, 1, 2, and 3, such that only two colors are used in the cells belonging to the same axis. This can be done by coloring the cells in the same x-axis with either the colors 0 and 1 or the colors 2 and 3, coloring the cells in the same y-axis with either the colors 0 and 2 or the colors 1 and 3, and coloring the cells in the same z-axis with either the colors 0 and 3 or the colors 1 and 2. Such a coloring is depicted in the left part of Figure 2.

Algorithm CRS-A randomly selects one out of the four colors and executes the greedy algorithm on the cells colored with the other three colors, ignoring (i.e., rejecting) all calls in cells colored with the selected color.

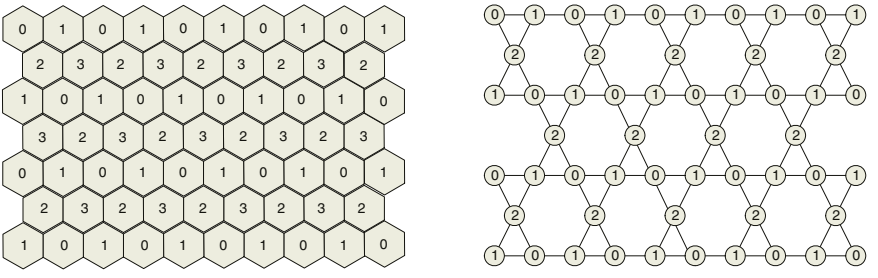


Fig. 2. The 4-coloring used by algorithm CRS-A and the corresponding subgraph of the interference graph induced by the nodes not colored with color 3.

Theorem 1. *Algorithm CRS-A in cellular networks of reuse distance* 2 *supporting one frequency is* 8/3*-competitive against oblivious adversaries.*

Proof. Let σ be a sequence of calls and denote by O the set of calls accepted by the optimal algorithm. Denote by σ' the set of calls in cells which are not colored with the color selected and by O' the set of calls the optimal algorithm would have accepted on input σ'. Clearly, $|O'|$ will be at least as large as the subset of O which belongs to σ'. Since the probability that the cell of a call in O is not colored with the color selected is 3/4, it is $\mathcal{E}[|O'|] \geq \frac{3}{4}|O|$. Now let B be the set of calls accepted by algorithm CRS-A, i.e., the set of calls accepted by the greedy algorithm when executed on sequence σ'. Observe that each call in O' either belongs in B or it is rejected because some other call is accepted. Furthermore, a call in $B \setminus O'$ can cause the rejection of at most two calls of O'. This implies that $|B| \geq |O'|/2$ which yields that the competitive ratio of algorithm CRS-A is

$$\frac{|O|}{\mathcal{E}[|B|]} \leq \frac{2|O|}{\mathcal{E}[|O'|]} \leq \frac{8}{3}.$$

□

The main advantage of algorithm CRS-A is that it uses only two random bits. In the next section we present simple on-line algorithms with improved competitive ratios that use slightly stronger random sources and work on networks with arbitrarily many frequencies.

3 Improved Algorithms

Algorithm CRS-A can be seen as an algorithm based on the "classify and randomly select" paradigm. It uses a coloring of the interference graph (not necessarily using the minimum possible number of colors) and a classification of the

colors. It starts by randomly selecting a color class (i.e., a set of colors) and then run the greedy algorithm in the cells colored with colors from this color class, ignoring (i.e., rejecting) calls in cells colored with colors not belonging to this class. Algorithm CRS-A uses a coloring of the interference graph with four colors 0, 1, 2, and 3, and the four color classes $\{0,1,2\}, \{0,1,3\}, \{0,2,3\}$, and $\{1,2,3\}$. Note that, in the previously known algorithms based on the "classify and randomly select" paradigm, color classes are singletons (e.g., [1], [10]).

The following simple lemma gives a sufficient condition for obtaining efficient on-line algorithms based on the "classify and randomly select" paradigm.

Lemma 1. *Consider a network with interference graph $G = (V, E)$ which supports w frequencies and let χ be a coloring of the nodes of V with the colors of a set X. If there exist ν sets of colors $s_0, s_1, ..., s_{\nu-1} \subseteq X$ and an integer λ such that*

- *each color of X belongs to at least λ different sets of the sets $s_0, s_1, ..., s_{\nu-1}$, and*
- *for $i = 0, 1, ..., \nu - 1$, each connected component of the subgraph of G induced by the nodes colored with colors in s_i is a clique,*

then there exists an on-line randomized call control algorithm for the network G which has competitive ratio ν/λ against oblivious adversaries.

Proof. Consider a network with interference graph G which supports w frequencies and the randomized on-line algorithm working as follows. The algorithm randomly selects one out of the ν color classes $s_0, ..., s_{\nu-1}$ and executes the greedy algorithm on the cells colored with colors of the selected class, rejecting all calls in cells colored with colors not in the selected class.

Let σ be a sequence of calls and let O be the set of calls accepted by the optimal algorithm on input σ. Assume that the algorithm selects the color class s_i. Let σ' be the sequence of calls in cells colored with colors in s_i and O' be the set of calls accepted by the optimal algorithm on input σ'. Also, we denote by B the set of calls accepted by the algorithm. First we can easily show that $|B| = |O'|$. Let G_j be a connected component of the subgraph of G induced by the nodes of G colored with colors in s_i. Let σ_j be the subsequence of σ' in cells corresponding to nodes of G_j. Clearly, any algorithm (including the optimal one) will accept at most one call of σ_j at each frequency. If the optimal algorithm accepts w calls, this means that the sequence σ_j has at least w calls and the greedy algorithm, when executed on σ', will accept w calls from σ_j (one call in each one of the available frequencies). If the optimal algorithm accepts $w' < w$ calls from σ_j, this means that σ_j contains exactly $w' < w$ calls and the greedy algorithm will accept them all in w' different frequencies. Since a call of σ_j is not constrained by a call in $\sigma_{j'}$ for $j \neq j'$, we obtain that $|B| = |O'|$.

The proof is completed by observing that the expected benefit of the optimal algorithm on input σ' over all possible sequences σ' defined by the random selection of the algorithm is $\mathcal{E}[|O'|] \geq \frac{\nu}{\lambda}|O|$, since, for each call in O, the probability that the color of its cell belongs to the color class selected is at least ν/λ. Hence, the competitive ratio of the algorithm against oblivious adversaries is

$$\frac{|O|}{\mathcal{E}[|B|]} = \frac{|O|}{\mathcal{E}[|O'|]} \leq \lambda/\nu.$$

□

Next, we present two randomized on-line algorithms for call control in cellular networks of reuse distance 2, namely CRS-B and CRS-C, which are also based on the "classify and randomly select" paradigm and achieve even better competitive ratios.

Consider a coloring of the cells with five colors $0, 1, 2, 3$, and 4 such that for each $i \in \{0, 1, 2, 3, 4\}$, and for each cell colored with color i, the two adjacent cells in the same x-axis are colored with colors $(i-1) \bmod 5$ and $(i+1) \bmod 5$, while the remaining four of its adjacent cells are colored with colors $(i+2) \bmod 5$ and $(i+3) \bmod 5$. Such a coloring is depicted in the left part of Figure 3. Also, define $s_i = \{i, (i+1) \bmod 5\}$, for $i = 0, 1, ..., 4$. Observe that, for each $i = 0, 1, ..., 4$, each pair of adjacent cells colored with the colors i and $(i+1) \bmod 5$ is adjacent to cells colored with colors $(i+2) \bmod 5$, $(i+3) \bmod 5$, and $(i+4) \bmod 5$, i.e., colors not belonging to s_i. Thus, the coloring χ together with the color classes s_i satisfy the conditions of Lemma 1 with $\nu = 5$ and $\lambda = 2$. We call CRS-B the algorithm that uses this coloring and works according to the "classify and randomly select" paradigm as in the proof of Lemma 1. We obtain the following.

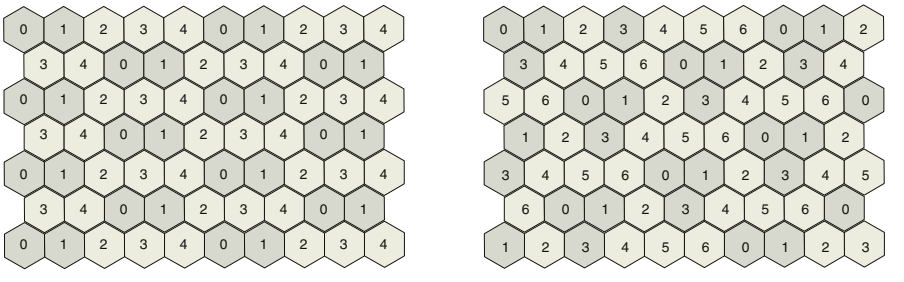


Fig. 3. The 5-coloring used by algorithm CRS-B and the 7-coloring used by algorithm CRS-C. The skewed cells are those colored with the colors in set s_0.

Theorem 2. *Algorithm CRS-B in cellular networks with reuse distance* 2 *is* 5/2*-competitive against oblivious adversaries.*

Now consider a coloring of the cells with seven colors $0, 1, ..., 6$ such that for each for each cell colored with color i (for $i = 0, ..., 6$), its two adjacent cells in the same x-axis are colored with the colors $(i-1) \bmod 7$ and $(i+1) \bmod 7$, while its two adjacent cells in the same z-axis are colored with colors $(i-3) \bmod 7$ and $(i+3) \bmod 7$. Such a coloring is depicted in the right part of Figure 3. Also, define $s_i = \{i, (i+1) \bmod 7, (i+3) \bmod 7\}$, for $i = 0, 1, ..., 6$. Observe that, for each $i = 0, 1, ..., 6$, each triangle of cells colored with the colors i, $(i+1) \bmod 7$, and $(i+3) \bmod 7$ is adjacent to cells colored with colors $(i+2) \bmod 7$, $(i+4) \bmod 7$,

$(i + 5) \bmod 7$, and $(i + 6) \bmod 7$, i.e., colors not belonging to s_i. Thus, the coloring χ together with the color classes s_i satisfy the conditions of Lemma 1 with $\nu = 7$ and $\lambda = 3$. We call CRS-C the algorithm that uses this coloring and works according to the "classify and randomly select" paradigm as in the proof of Lemma 1. We obtain the following.

Theorem 3. *Algorithm CRS-C in cellular networks with reuse distance* 2 *is* $7/3$*-competitive against oblivious adversaries.*

The two algorithms above (CRS-B and CRS-C) make use of a random source which equiprobably selects one out of an odd number of distinct objects. If we only have a number of fair coins (random bits) available, we can design algorithms with small competitive ratios by combining the algorithms above. For example, using 6 random bits, we may construct the following algorithm. We use integers $0, 1, ..., 63$ to identify each of the 63 outcomes of the 6 random bits. For an outcome $i \in \{0, ..., 49\}$, the algorithm executes algorithm CRS-B using color class $s_{i \bmod 5}$, and for an outcome $i \in \{50, ..., 63\}$, the algorithm executes algorithm CRS-C using color class $s_{(i-50) \bmod 7}$. It can be easily seen that this algorithm has competitive ratio $32/13 \approx 2.462$ against oblivious adversaries, since its expected benefit is at least $50/64 \cdot 2/5 + 14/64 \cdot 3/7 = 13/32$ times the optimal benefit. Similarly, using 8 random bits, we obtain an on-line algorithm with competitive ratio $64/27 \approx 2.37$. We can generalize this idea, and, for sufficiently small $\epsilon > 0$, we can construct an algorithm which uses $t = O(\log 1/\epsilon)$ random bits, and, on $2^t \bmod 7$ of the 2^t outcomes, it does nothing, while the rest of the outcomes are assigned to executions of algorithm CRS-C. In this way, we obtain the following.

Corollary 1. *For any* $\epsilon > 0$*, there exists an on-line randomized call-control algorithm for cellular networks with reuse distance* 2 *that uses* $O(\log 1/\epsilon)$ *random bits and has competitive ratio at most* $7/3 + \epsilon$ *against oblivious adversaries.*

4 Cellular Networks with Reuse Distance $k > 2$

For cellular networks with reuse distance $k > 2$, we present algorithm CRS-k which is based on the "classify and randomly select" paradigm. Algorithm CRS-k uses the following coloring of the interference graph of a cellular network with reuse distance k. Cells are colored with the colors $0, 1, ..., 3k^2 - 3k$ such that for any cell colored with color i, its adjacent cells in the x-axis are colored with colors $(i-1) \bmod (3k^2-3k+1)$ and $(i+1) \bmod (3k^2-3k+1)$, while its adjacent cells in the z-axis are colored with colors $(i - 3(k-1)^2) \bmod (3k^2 - 3k + 1)$ and $(i + 3(k-1)^2) \bmod (3k^2 - 3k + 1)$.

For odd k, for $i = 0, 1, ..., 3k^2 - 3k$, the color class s_i contains the following colors. For $j = 0, 1, ..., \frac{k-1}{2}$, it contains the colors $(i + 3j(k-1)^2 - j) \bmod (3k^2 - 3k + 1)$, ..., $(i + 3j(k-1)^2 + \frac{k-1}{2}) \bmod (3k^2 - 3k + 1)$, and for $j = \frac{k+1}{2}, ..., k-1$, it contains the colors $(i + \frac{3(k-1)^3}{2} + 3(j - \frac{k-1}{2})(k-1)^2 - \frac{k-1}{2}) \bmod (3k^2 - 3k + 1)$, ..., $(i + \frac{3(k-1)^3}{2} + 3(j - \frac{k-1}{2})(k-1)^2 + k - 1 - j) \bmod (3k^2 - 3k + 1)$.

For even k, for $i = 0, 1, ..., 3k^2 - 3k$, the color class s_i contains the following colors. For $j = 0, 1, ..., \frac{k}{2} - 1$, it contains the colors $(i + 3j(k-1)^2 - j) \bmod (3k^2 - 3k + 1), ..., (i + 3j(k-1)^2 + \frac{k}{2}) \bmod (3k^2 - 3k + 1)$, and for $j = \frac{k}{2}, ..., k-1$, it contains the colors $(i + 3(\frac{k}{2} - 1)(k-1)^2 + 3(j - \frac{k}{2} + 1)(k-1)^2 - \frac{k}{2} + 1) \bmod (3k^2 - 3k + 1)$, ..., $(i + 3(\frac{k}{2} - 1)(k-1)^2 + 3(j - \frac{k}{2} + 1)(k-1)^2 + k - 1 - j) \bmod (3k^2 - 3k + 1)$.

Note that, for $k = 2$, we obtain the coloring used by algorithm CRS-C. Examples of the coloring for $k = 3$ and $k = 4$ as well as the cells colored with colors from the color class s_0 are depicted in Figure 4.

We can show the following two lemmas. Formal proofs will be given in the final version of the paper.

Lemma 2. *Let $k > 2$ and G be the interference graph of a cellular network of distance reuse k. Consider the coloring of G used by algorithm CRS-k and the color classes s_i, for $i = 0, 1, ..., 3k^2 - 3k$. For any color j such that $0 \leq j \leq 3k^2 - 3k$, the number of different color classes s_i that color j belongs to is $\frac{3k^2}{4}$ if k is even, and $\frac{3k^2+1}{4}$ if k is odd.*

Lemma 3. *Let $k > 2$ and G be the interference graph of a cellular network of distance reuse k. Consider the coloring of G used by algorithm CRS-k and the color classes s_i, for $i = 0, 1, ..., 3k^2 - 3k$. For $i = 0, 1, ..., 3k^2 - 3k$, each connected component of the subgraph of G induced by the nodes of G colored with colors in s_i is a clique.*

Thus, the colorings and the color classes described satisfy the condition of Lemma 1 with $\lambda = 3k^2 - 3k + 1$ and $\nu = \frac{3k^2}{4}$ if k is even, and $\nu = \frac{3k^2+1}{4}$ if k is odd. By Lemma 1, we obtain the following.

Theorem 4. *The competitive ratio against oblivious adversaries of algorithm CRS-k in cellular networks with reuse distance $k \geq 2$ is $4\left(1 - \frac{3k-1}{3k^2}\right)$ if k is even, and $4\left(1 - \frac{3k}{3k^2+1}\right)$ if k is odd.*

Algorithm CRS-k in cellular networks of reuse distance k uses a random source which equiprobably selects one among $3k^2 - 3k + 1$ distinct objects. By applying ideas we used in the previous section, we can achieve similar competitiveness bounds by algorithms that use random bits.

Corollary 2. *For any $\epsilon > 0$, there exists an on-line randomized call-control algorithm for cellular networks of reuse distance k, that uses $O(\log 1/\epsilon + \log k)$ random bits and has competitive ratio at most $4\left(1 - \frac{3k-1}{3k^2}\right) + \epsilon$ if k is even, and $4\left(1 - \frac{3k}{3k^2+1}\right) + \epsilon$ if k is odd, against oblivious adversaries.*

5 Lower Bounds

In previous work ([4], implicitly in [10]), it has been observed that no deterministic on-line call control algorithm can have a competitive ratio better than 3 against off-line adversaries. We can easily extend this lower bound and obtain

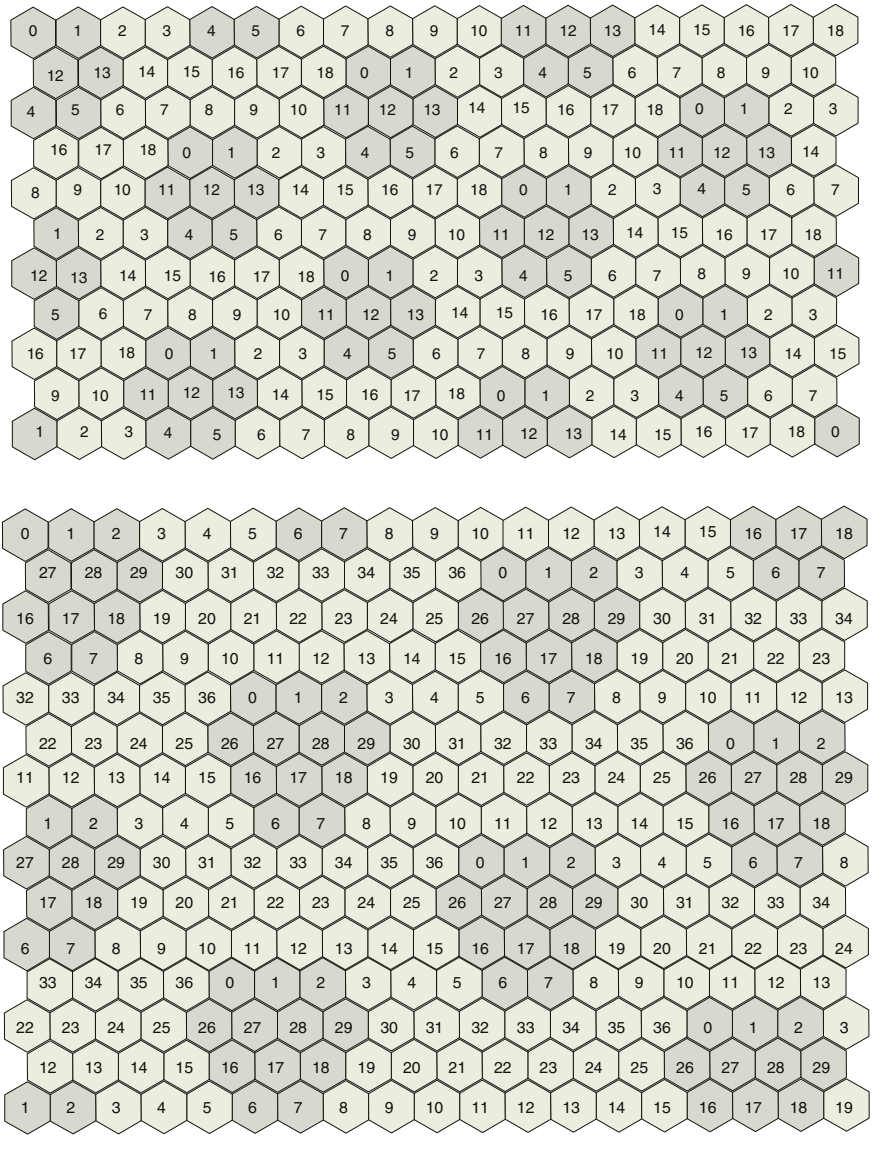


Fig. 4. Examples of the colorings used by the algorithms CRS-3 and CRS-4. The skewed cells are those colored with the colors in set s_0.

lower bounds of 4 and 5 on the competitiveness of deterministic on-line call control algorithms in cellular networks of reuse distance $k \in \{3, 4, 5\}$ and $k \geq 6$, respectively.

Hence, the randomized algorithms presented in the previous section significantly beat the lower bound on the competitiveness of deterministic algorithms.

In what follows, using the Minimax Principle [13] (see also [9]), we prove a lower bound on the competitive ratio, against oblivious adversaries, of any randomized algorithm in cellular networks with reuse distance $k \geq 5$. Again, we consider networks that support one frequency; our lower bound can be trivially extended to networks that support multiple frequencies. In our proof, we use the following lemma.

Lemma 4 (Minimax Principle [9]). *Given a probability distribution $\mathcal{P}$ over sequences of calls σ, denote by $\mathcal{E}_{\mathcal{P}}[B_A(\sigma)]$ and $\mathcal{E}_{\mathcal{P}}[B_{OPT}(\sigma)]$ the expected benefit of a deterministic algorithm A and the optimal off–line algorithm on sequences of calls generated according to $\mathcal{P}$. Define the competitiveness of A under $\mathcal{P}$, $c_A^{\mathcal{P}}$ to be such that*

$$c_A^{\mathcal{P}} = \frac{\mathcal{E}_{\mathcal{P}}[B_{OPT}(\sigma)]}{\mathcal{E}_{\mathcal{P}}[B_A(\sigma)]}.$$

Let A_R be a randomized algorithm. Then, the competitiveness of A under $\mathcal{P}$ is a lower bound on the competitive ratio of A_R against an oblivious adversary, i.e. $c_A^{\mathcal{P}} \leq c_A$.

Our lower bound is the following.

Theorem 5. *No randomized call–control algorithm in cellular networks with distance reuse $k \geq 5$ can be better than $25/12$–competitive against an oblivious adversary.*

Proof. Consider a cellular network with reuse distance 5 and ten cells v_0, $v_{A,1}$, $v_{A,2}$, $v_{B,1}$, $v_{B,2}$, $v_{B,3}$, $v_{C,1}$, $v_{C,2}$, $v_{C,3}$, and $v_{C,4}$ as shown in Figure 5. We will prove that there exists an adversary $\mathcal{ADV}$ that produces calls in these cells according to a probability distribution $\mathcal{P}$ in such way that no deterministic algorithm can be better than $25/12$–competitive under $\mathcal{P}$ even if it knows the probability distribution $\mathcal{P}$ in advance.

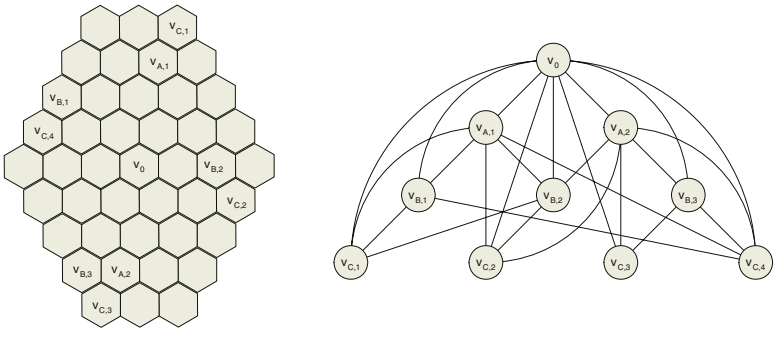


Fig. 5. The cellular network of reuse distance 5 used in the proof of Theorem 9 and the subgraph of the interference graph induced by the nodes corresponding to the ten cells v_0, $v_{A,1}$, $v_{A,2}$, $v_{B,1}$, $v_{B,2}$, $v_{B,3}$, $v_{C,1}$, $v_{C,2}$, $v_{C,3}$, and $v_{C,4}$.

We define the probability distribution $\mathcal{P}$ as follows. First, the adversary produces a call in the cell v_0. Then, it

- either stops, with probability 1/2,
- or does the following, with probability 1/2. It presents two calls, one in the cell $v_{A,1}$ and one in the cell $v_{A,2}$, and
  - either stops, with probability 1/3,
  - or does the following with probability 2/3. It presents three calls, one in the cell $v_{B,1}$, one in the cell $v_{B,2}$, and one in the cell $v_{B,3}$, and
    - either stops, with probability 1/4,
    - or does the following, with probability 3/4. It presents four calls, one in the cell $v_{C,1}$, one in the cell $v_{C,2}$, one in the cell $v_{C,3}$, and one in the cell $v_{C,4}$, and then stops.

Clearly, the benefit of the optimal off-line algorithm is the number of calls presented by the adversary in the last step before stopping the sequence. Thus, the expected benefit of the optimal off–line algorithm on sequences of calls generated according to $\mathcal{P}$ is

$$\mathcal{E}_{\mathcal{P}}[B_{OPT}(\sigma)] = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{2} \cdot \frac{1}{3} + 3 \cdot \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{1}{4} + 4 \cdot \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} = \frac{25}{12}.$$

Let A be a deterministic call control algorithm that runs on the calls produced by $\mathcal{ADV}$. First, we observe that no algorithm could gain by accepting one of the two calls presented in cells $v_{A,1}$ and $v_{A,2}$ or by accepting one or two of the three calls presented in cells $v_{B,1}$, $v_{B,2}$ and $v_{B,3}$. Hence, we may assume that the algorithm either accepts all calls presented at a step or rejects them all.

Consider t executions of the algorithm on t sequences produced according to the probability distribution $\mathcal{P}$. Let q_0 be the number of executions in which A accepts the call produced in cell v_0, q_1 the number of executions in which A accepts both calls in cells $v_{A,1}$ and $v_{A,2}$, and q_2 the number of executions in which the algorithm accepts the three calls in cells $v_{C,1}$, $v_{C,2}$, and $v_{C,3}$.

The expected number of executions in which the algorithm does not accept the call in cell v_0 and the adversary produces calls in cells $v_{A,1}$ and $v_{A,2}$ is $\frac{1}{2}(t-q_0)$. Hence, the expected number of executions in which the algorithm does not accept the calls in cells v_0, $v_{A,1}$, and $v_{A,2}$ and the adversary produces calls in cells $v_{B,1}$, $v_{B,2}$, and $v_{B,3}$ is $\frac{2}{3}\left(\frac{1}{2}(t-q_0)-q_1\right)$ and the expected number of executions in which the algorithm does not accept the calls in cells v_0, $v_{A,1}$, $v_{A,2}$, $v_{B,1}$, $v_{B,2}$, and $v_{B,3}$ and the adversary produces calls in cells $v_{C,1}$, $v_{C,2}$, $v_{C,3}$, and $v_{C,4}$ is $\frac{3}{4}\left(\frac{2}{3}\left(\frac{1}{2}(t-q_0)-q_1\right)-q_2\right)$. Thus,

$$\mathcal{E}_{\mathcal{P}}[B_A(\sigma)] \leq \frac{q_0 + 2q_1 + 3q_2 + 4\frac{3}{4}\left(\frac{2}{3}\left(\frac{1}{2}(t-q_0)-q_1\right)-q_2\right)}{t} = 1$$

and $c_A^{\mathcal{P}} \geq 25/12$. By Lemma 4, we obtain that this is a lower bound on the competitiveness of any randomized algorithm against oblivious adversaries. □

References

1. B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosen. Competitive On–line Competitive Algorithms for Call Admission in Optical Networks. *Algorithmica*, Vol. 31(1), pp. 29-43, 2001.
2. B. Awerbuch, Y. Bartal, A. Fiat, A. Rosen. Competitive Non–Preemptive Call Control. In *Proccedings of the 5th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA '94)*, pp. 312–320, 1994.
3. Y. Bartal, A. Fiat, and S. Leonardi. Lower Bounds for On–line Graph Problems with Applications to On–line Circuit and Optical Routing. In *Proc. of the 28th Annual ACM Symposium on Theory of Computing (STOC '96)*, 1996.
4. I. Caragiannis, C. Kaklamanis, and E. Papaioannou. Efficient On–Line Frequency Allocation and Call Control in Cellular Networks. *Theory of Computing Systems*, Vol. 35, pp. 521-543, 2002.
5. T. Erlebach and K. Jansen. The Maximum Edge-Disjoint Paths Problem in Bidirected Trees. *SIAM Journal on Discrete Mathematics*, Vol. 14(3), pp. 326-355, 2001.
6. W.K. Hale. Frequency Assignment: Theory and Applications. In *Proceedings of the IEEE*, 68(12), pp. 1497–1514, 1980.
7. J. Janssen, D. Krizanc, L. Narayanan, and S. Shende. Distributed On–Line Frequency Assignment in Cellular Networks. *Journal of Algorithms*, Vol. 36(2), pp. 119-151, 2000.
8. S. Leonardi, A. Marchetti–Spaccamela, A. Prescuitti, and A. Rosen. On–line Randomized Call–Control Revisited. In *Proccedings of the 9th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA '98)*, pp. 323–332, 1998.
9. R. Motwani and B. Raghavan. Randomized Algorithms. *Cambridge University Press*, 1995.
10. G. Pantziou, G. Pentaris, and P. Spirakis. Competitive Call Control in Mobile Networks. *Theory of Computing Systems*, Vol. 35(6), pp. 625-639, 2002.
11. D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of Association of Computing Machinery* 28, pp. 202–208, 1985.
12. P.-J. Wan and L. Liu. Maximal Throughput in Wavelength-Routed Optical Networks. Multichannel Optical Networks: Theory and Practice, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, AMS, Vol. 46, pp. 15-26, 1998.
13. A. C. Yao. Probabilistic Computations: Towards a Unified Measure of Complexity. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science (FOCS '77)*, pp. 222–227, 1977.

Fractional and Integral Coloring of Locally-Symmetric Sets of Paths on Binary Trees*

Ioannis Caragiannis[1], Christos Kaklamanis[1], Pino Persiano[2], and Anastasios Sidiropoulos[3]

[1] Computer Technology Institute and Dept. of Computer Engineering and Informatics University of Patras, 26500 Rio, Greece {caragian,kakl}@cti.gr

[2] Dipartimento di Informatica ed Applicazioni Università di Salerno, 84081 Baronissi, Italy giuper@dia.unisa.it

[3] Computer Science and Artificial Intelligence Laboratory Massachusetts Institute of Technology 200 Technology Square, Cambridge, MA 02139, USA tasos@mit.edu

Abstract. Motivated by the problem of allocating optical bandwidth in tree–shaped WDM networks, we study the fractional path coloring problem in trees. We consider the class of locally-symmetric sets of paths on binary trees and prove that any such set of paths has a fractional coloring of cost at most $1.367L$, where L denotes the load of the set of paths. Using this result, we obtain a randomized algorithm that colors any locally-symmetric set of paths of load L on a binary tree (with reasonable restrictions on its depth) using at most $1.367L + o(L)$ colors, with high probability.

1 Introduction

Let $T(V, E)$ be a bidirected tree, i.e., a tree with each edge consisting of two opposite directed edges. Let $\mathcal{P}$ be a set of directed paths on T. The *path coloring problem* (or integral path coloring problem) is to assign colors to paths in $\mathcal{P}$ so that no two paths that share a directed edge of T are assigned the same color and the total number of colors used is minimized. The problem has applications to WDM (Wavelength Division Multiplexing) routing in tree–shaped all–optical networks. In such networks, communication requests are considered as ordered transmitter–receiver pairs of network nodes. WDM technology establishes communication by finding transmitter–receiver paths and assigning a wavelength to each path, so that no two paths going through the same fiber

* This work was partially funded by the European Union under IST FET Project ALCOM–FT, IST FET Project CRESCCO and RTN Project ARACNE.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 81–94, 2004.
© Springer-Verlag Berlin Heidelberg 2004

are assigned the same wavelength. Since state–of–the–art technology [10] allows for a limited number of wavelengths, the important engineering question to be solved is to establish communication so that the total number of wavelengths used is minimized.

The path coloring problem in trees has been proved to be NP–hard in [5], thus the work on the topic mainly focuses on the design and analysis of approximation algorithms. Known results are expressed in terms of the load L of $\mathcal{P}$, i.e., the maximum number of paths that share a directed edge of T. Note that, for any set of paths, its load is a lower bound on the optimal number of colors. An algorithm that assigns at most $2L$ colors to any set of paths is implicit in the work of Raghavan and Upfal [9]. The best known upper bound for arbitrary trees is $5L/3$ [6]. A randomized $(1.613 + o(1))$–approximation algorithm for trees of bounded degree is presented in [2].

The path coloring problem is still NP–hard when the input instances are restricted to sets of paths on binary trees [5]. A randomized path coloring algorithm for binary trees is presented in [1]. This algorithm colors any set of paths of load L on a binary tree (with some restrictions on its depth) using at most $7L/5+o(L)$ colors. In [8], it is proved that there are sets of paths on binary trees whose optimal path colorings require at least $5L/4$ colors.

The path coloring problem is still NP–hard when the input instances are restricted to symmetric sets of paths on binary trees [3]. A set of paths $\mathcal{P}$ is called symmetric if it can be partitioned into disjoint pairs of symmetric paths, i.e., (u, v) and (v, u). Symmetric sets of paths are important since many services which are supported by WDM networks (or are expected to be supported in the future) require bidirectional reservation of bandwidth. Algorithms that color any symmetric set of paths of load L on a tree (not necessarily binary) with $3L/2$ colors can be obtained by considering each pair of symmetric paths as an undirected one and then applying an algorithm for coloring undirected paths on undirected trees [5, 9]. For binary trees, the randomized algorithm presented in [1] gives the best known upper bound in this case. In [3] it is also proved that there are symmetric sets of paths on binary trees whose optimal path colorings require at least $5L/4$ colors.

The *fractional path coloring problem* is a natural relaxation of path coloring. Given a set of paths on a tree, a solution of the path coloring problem is also a solution for the fractional path coloring problem with cost equal to the number of colors in the solution of path coloring. Furthermore, the lower bound proofs in [3] and [8] still hold for fractional path coloring. Finding a fractional path coloring of optimal cost can be solved in polynomial time in trees of bounded degree [2]. Fractional path colorings are important since they can be used to obtain path colorings using a number of colors which is provably close to the cost of the fractional solution by applying randomized rounding techniques [2, 7]. Moreover, it is interesting to prove upper bounds on the cost of optimal fractional path colorings in terms of the load, since they give insight to the path coloring problem. In [2], the result of [1] was extended to prove that any set of paths of load L on a binary tree has a fractional path coloring of cost $7L/5$.

In this paper, we consider locally-symmetric sets of paths on binary trees. A set of paths $\mathcal{P}$ on a tree is called locally-symmetric if, for any two nodes u and v of the tree with distance at most 2, the number of paths coming from v and going to u equals the number of paths coming from u and going to v. Clearly, the class of locally-symmetric sets of paths on a tree T contain the class of symmetric sets of paths on T. We prove that any locally-symmetric set of paths of load L on a binary tree has a fractional path coloring of cost at most $1.367L$. This fractional path coloring has some additional nice properties, and, using this result and techniques of [1], we obtain a randomized algorithm that colors any locally-symmetric set of paths of load L on a binary tree (with some restrictions on its depth) using $1.367L + o(L)$ colors, with high probability. Since the load of a set of paths is a lower bound on the minimum number of colors sufficient for coloring it, our algorithm is an $(1.367 + o(1))$-approximation algorithm.

The rest of the paper is structured as follows. In Section 2 we give formal definitions for the (fractional) path coloring problem. In Section 3 we discuss some properties of locally-symmetric sets of paths which are useful in the analysis of our algorithms. We describe the fractional path coloring algorithm in Section 4 and present its analysis in Section 5. We discuss the extension of the result for fractional path coloring to path coloring in Section 6.

2 Fractional Path Colorings

The graph coloring problem can be considered as finding an integral covering of the vertices of a graph by independent sets of unit weight so that the total weight (or cost) is minimized. Given a graph $G = (V, E)$, this means solving the following integer linear program:

$$\begin{array}{rll} \text{minimize} & \sum_{I \in \mathcal{I}} x(I) & \\ \text{subject to} & \sum_{I \in \mathcal{I}: v \in I} x(I) \geq 1 & v \in V \\ & x(I) \in \{0, 1\} & I \in \mathcal{I} \end{array}$$

where $\mathcal{I}$ denotes the set of the independent sets of G.

This formulation has a natural relaxation into the following linear program:

$$\begin{array}{rll} \text{minimize} & \sum_{I \in \mathcal{I}} \bar{x}(I) & \\ \text{subject to} & \sum_{I \in \mathcal{I}: v \in I} \bar{x}(I) \geq 1 & v \in V \\ & \bar{x}(I) \geq 0 & I \in \mathcal{I} \end{array}$$

The corresponding combinatorial problem is called the *fractional coloring problem*. If $\bar{x}$ is a valid weight assignment over the independent sets of the graph G, we call it a *fractional coloring* of G.

Given a set of paths $\mathcal{P}$ on a graph G, we define an *independent set of paths* as a set of pairwise edge–disjoint paths. Equivalently, we may think of the conflict graph of $\mathcal{P}$ which is the graph having a node for each path of $\mathcal{P}$ and edges between any two nodes corresponding to conflicting (i.e., not edge–disjoint) paths; an independent set of paths corresponds to an independent set on the conflict graph.

The fractional path coloring problem is defined as the fractional coloring problem on the conflict graph.

In [2] it is proved that optimal fractional path colorings in bounded–degree trees can be computed by solving a linear program of polynomial (in terms of the load of the set of paths and the size of the tree) size. These techniques can be extended to graphs of bounded degree and bounded treewidth. A polynomial time algorithm for computing optimal fractional path colorings in rings is implicit in [7].

Although computing an optimal fractional path coloring of a locally-symmetric set of paths $\mathcal{P}$ on a binary tree can be done in polynomial time, in this paper we are interested in exploring the relation of the cost of the optimal fractional path coloring with the load of $\mathcal{P}$.

3 Properties of Locally-Symmetric Sets of Paths

In this section we give some useful properties of locally-symmetric sets of paths on binary trees. Let T be a binary tree. Without loss of generality, we assume that all nodes of T have degree either 1 or 3. We denote by r (root) a leaf node of T. Starting from r, we assign labels to the nodes of the tree by performing a breadth-first-search (r is assigned 0). For each non-leaf node v, we will denote by $p(v)$ the parent of v, and by $l(v)$ and $r(v)$ the left and right child of v, respectively.

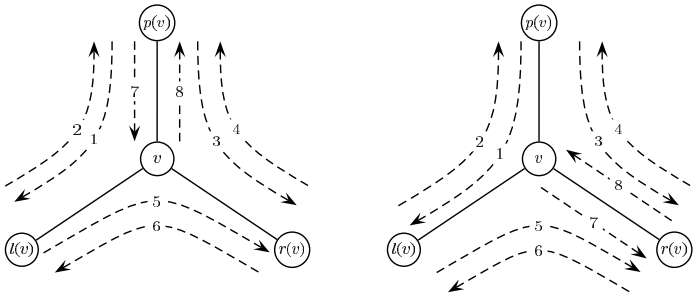
Let $\mathcal{P}$ be a locally-symmetric set of paths. We may assume that $\mathcal{P}$ is *normal*, i.e., it satisfies the following properties:

- It has full load L at each directed edge.
- For every node u, the paths that originate or terminate at u, appear on only one of the three edges adjacent to u.

If the initial set of paths is not normal, we can transform it to a normal one by first adding pairs of symmetric single–hop paths on the edges of the tree which are not fully–loaded, and then, for each non-leaf node v, by merging paths terminating at v with paths originating at v that do not traverse the same edge in opposite directions. It can be easily verified that this can be done in such a way that the resulting set of paths is locally-symmetric.

Claim. Let $\mathcal{P}$ be a locally-symmetric set of paths on a binary tree T with load L. There exists a normal locally-symmetric set of paths $\mathcal{P}'$ such that if $\mathcal{P}'$ has a (fractional) coloring of cost c, then $\mathcal{P}$ has a (fractional) coloring of cost at most c.

We partition the set of paths of $\mathcal{P}$ that go through v to the following disjoint subsets: the set M_v^1 of the paths that come from $p(v)$ and go to $l(v)$, the set M_v^2 of the paths that come from $l(v)$ and go to $p(v)$, the set M_v^3 of the paths that come from $p(v)$ and go to $r(v)$, the set M_v^4 of the paths that come from $r(v)$ and go to $p(v)$, the set M_v^5 of the paths that come from $l(v)$ and go to $r(v)$, and the set M_v^6 of the paths that come from $r(v)$ and go to $l(v)$. Since $\mathcal{P}$ is normal,



(a) Scenario I (b) Scenario II

Fig. 1. The two cases for paths touching a non-leaf node v. Numbers represent groups of paths (number 1 implies the set of paths M_v^1, etc.).

we only need to consider two cases for the set P_v of paths of $\mathcal{P}$ touching node v. These cases are depicted in Figure 1.

- **Scenario I:** The paths of P_v originating from or terminating at v touch node $p(v)$. We denote by M_v^7 the set of paths that come from $p(v)$ and stop at v and by M_v^8 the set of paths that originate from v and go to $p(v)$.
- **Scenario II:** The paths of P_v originating from or terminating at v touch a child node of v (wlog $r(v)$). We denote by M_v^7 the set of paths that originate from v and go to $r(v)$ and by M_v^8 the set of paths that come from $r(v)$ and stop at v.

Claim. Let $\mathcal{P}$ be a normal locally-symmetric set of paths on a binary tree T with load L. Let v be a non-leaf node of T and P_v be the subset of P that contains the paths touching v.

- If P_v belongs to Scenario I, then $|M_v^1| = |M_v^2| = |M_v^3| = |M_v^4| \leq L/2$, $|M_v^5| = |M_v^6| = L - |M_v^1|$, and $|M_v^7| = |M_v^8| = L - 2|M_v^1|$.
- If P_v belongs to Scenario II, then $|M_v^1| = |M_v^2| \geq L/2$, $|M_v^3| = |M_v^4| = |M_v^5| = |M_v^6| = L - |M_v^1|$, and $|M_v^7| = |M_v^8| = 2|M_v^1| - L$.

4 The Algorithm

In this section we present our fractional path coloring algorithm. It uses a parameter $D \in [2/3, \frac{2+\sqrt{2}}{4}]$.

Let T be a binary tree and $\mathcal{P}$ a locally-symmetric set of paths on T. We denote by $\mathcal{I}$ the set of all independent sets of paths in $\mathcal{P}$. Given an edge e, we denote by $\mathcal{I}_e^S$ the set of independent sets of $\mathcal{I}$ that contain exactly one path traversing e in some direction and by $\mathcal{I}_e^D$ the set of independent sets of $\mathcal{I}$ that contain two paths traversing e (in opposite directions).

Definition 1 (Property 1). *A weight assignment x on the independent sets of $\mathcal{I}$ satisfies Property 1 at a node v if*

- *Property 1 is satisfied on each node u which has label smaller than the label of v, and*
- *either v is a leaf different from the root or for any path p traversing an edge e between v and a child of v in some direction, it is*

$$\sum_{I \in \mathcal{I} \ : p \in I} x(I) = 1 - D.$$

Definition 2 (Property 2). *A weight assignment x on the independent sets of $\mathcal{I}$ satisfies Property 2 at a node v if*

- *Property 2 is satisfied on each node u which has label smaller than the label of v, and*
- *either v is a leaf different from the root or for any two paths p, q traversing an edge e between v and a child of v in opposite directions, it is*

$$\sum_{I \in \mathcal{I} \ : p,q \in I} x(I) = \frac{D}{L}.$$

Definition 3 (Property 3). *A weight assignment x on the independent sets of $\mathcal{I}$ satisfies Property 3 at a node v if for any independent set I of $\mathcal{I}$ that contains at least one path touching neither v nor nodes of label smaller than v's, it is $x(I) = 0$.*

Definition 4 (Property 4). *A weight assignment x on the independent sets of $\mathcal{I}$ satisfies Property 4 if*

$$\sum_{I \in \mathcal{I}} x(I) \leq \frac{4D^2 - 4D + 4}{3D} L.$$

Our algorithm assigns weights x to the independent sets of $\mathcal{I}$ such that x is a valid fractional coloring of small cost.

First, the algorithm performs the following initialization step:

- For each independent set $I \in \mathcal{I}$ consisting only of one path that terminates at or originates from r, it sets $x_r(I) = 1 - D$.
- For each independent set $I \in \mathcal{I}$ consisting only of two opposite directed paths one terminating at and the other originating from r, it sets $x_r(I) = D/L$.
- For any other independent set $I \in \mathcal{I}$, it sets $x_r(I) = 0$.

Then, for $i = 1, \ldots, n-1$ (where n is the number of nodes of the tree) , the algorithm executes the procedure Fract-Color on node v with label i.

The procedure Fract-Color at a node v takes as input the set P_v of paths touching node v and a non-negative weight assignment $x_{p(v)}$ which satisfies Properties 1, 2 and 3 at node $p(v)$ and Property 4. If v is a leaf, the procedure Fract-Color sets the weight assignment x_v equal to $x_{p(v)}$ and stops. Otherwise, it computes a new non-negative weight assignment x_v which satisfies Properties 1,

2 and 3 at node v and Property 4. It also sets the weight assignments x_u for all the nodes u with label smaller than v's equal to x_v.

Finally, the algorithm outputs a weight assignment $x = x_u$ where u is the node with label $n-1$.

Fract-Color will be precisely described in the next section. For now, we use its input-output specification to show inductively that the algorithm produces a weight assignment x which satisfies Properties 1, 2 and 3 at each node of T and Property 4. Clearly, after the initialization step, the assignment x_r is non-negative and satisfies Properties 1, 2 and 3 at node r. Also, under weight assignment x_r, there are $2L$ independent sets with weight $1-D$, L^2 independent sets with weight D/L while all other independent sets have zero weight. Thus, Property 4 is satisfied since

$$\sum_{I\in\mathcal{I}} x_r(I) = (2-D)L < \frac{4D^2-4D+4}{3D}L.$$

Assuming that the execution of Fract-Color at a node u with label i has produced a non-negative weight assignment x_u that satisfies Properties 1, 2 and 3 at u and Property 4, we can easily show that the execution of Fract-Color at a node v with label $i+1$ creates a non-negative weight assignment x_v that satisfies Properties 1, 2 and 3 at node v and Property 4. By the description of Fract-Color, we just have to show that the weight assignment x_u which is part of the input of the execution of Fract-Color at node v satisfies Properties 1, 2 and 3 at node $p(v)$ and Property 4. Property 4 is obviously satisfied. Since either $p(v) = u$ or $p(v)$ has a label smaller than u's, x_u satisfies Properties 1, 2 and 3 at $p(v)$.

Let p be a path traversing an edge e between some node v and a child of v in some direction. Note that p is contained only in independent sets of the disjoint sets I_e^S and I_e^D. Furthermore, x satisfies Properties 1, 2 and 3 at node v and there are L paths that traverse the edge e in opposite direction to p. We obtain that

$$\sum_{I\in\mathcal{I}:p\in I} x(I) = \sum_{I\in I\ :p\in I} x(I) + \sum_{I\in I\ :p\in I} x(I) = 1.$$

Also, x is non-negative. Thus, the algorithm specified above finds a fractional coloring x of the paths of $\mathcal{P}$ of cost at most $\frac{4D^2-4D+4}{3D}L$. In the following section, we describe the procedure Fract-Color and prove that it works correctly (i.e., as specified above) provided that $D \in [2/3, \frac{2+\sqrt{2}}{4}]$. Substituting $D = \frac{2+\sqrt{2}}{4}$, we obtain the following theorem.

Theorem 1. *For any locally-symmetric set of paths $\mathcal{P}$ of load L on a binary tree T, there exists a fractional coloring of cost at most* $1.367L$.

5 The Procedure Fract-Color

Let T be a binary tree and $\mathcal{P}$ a normal locally-symmetric set of paths on T. We show how the procedure Fract-Color works when executed at a non–leaf

node v. Let P_v be the set of paths of $\mathcal{P}$ touching node v. We have to consider two cases: one for Scenario I and one for Scenario II. Due to lack of space, we present only the case of Scenario I here.

We denote by I_v the set of independent sets of $\mathcal{I}$ that contain only paths touching nodes with label smaller than v. We denote by I_v^0 the set of independent sets in I_v that do not contain paths traversing the edge between v and its parent in some direction. We denote by I_v^i for $i = 1, 2, 3, 4, 7, 8$, the set of independent sets of I_v which contain a path p of M_v^i and no path traversing the edge between v and its parent in opposite direction to p, and by I_v^{ij} the set of independent sets of $\mathcal{I}$ which contain a path of M_v^i and a path of M_v^j, such that paths of M_v^i traverse the edge between v and its parent in opposite direction than paths in M_v^j.

When executed at v, Fract-Color takes as input the set P_v and a non-negative weight assignment $x_{p(v)}(I)$ on the independent sets of $\mathcal{I}$ that satisfies Properties 1, 2, and 3 at $p(v)$ and Property 4. Fract-Color computes a non-negative weight assignment $x_v(I)$ on the independent sets of $\mathcal{I}$ that satisfies Properties 1, 2, and 3 at v and Property 4. This is done in the following way:

- For each independent set I' of I_v^1 and I_v^4 and each path p of M_v^5, we set to $x_v(I) = \frac{\alpha}{L-|M^1|} x_{p(v)}(I')$ the weight of the independent set consisting of the paths of I' and path p and $x_v(I') = (1-\alpha)x_{p(v)}(I')$.
- For each independent set I' of I_v^2 and I_v^3 and each path p of M_v^6, we set to $x_v(I) = \frac{\alpha}{L-|M^1|} x_{p(v)}(I')$ the weight of the independent set consisting of the paths of I' and path p and $x_v(I') = (1-\alpha)x_{p(v)}(I')$.
- For each independent set I' of I_v^{14} and each path p of M_v^5, we set to $x_v(I) = \frac{\beta}{L-|M^1|} x_{p(v)}(I')$ the weight of the independent set consisting of the paths of I' and path p and $x_v(I') = (1-\beta)x_{p(v)}(I')$.
- For each independent set I' of I_v^{23} and each path p of M_v^6, we set to $x_v(I) = \frac{\beta}{L-|M^1|} x_{p(v)}(I')$ the weight of the independent set consisting of the paths of I' and path p and $x_v(I') = (1-\beta)x_{p(v)}(I')$.
- For each independent set I' of I_v^{18} and I_v^{47} and each path p of M_v^5, we set to $x_v(I) = \frac{\gamma}{L-|M^1|} x_{p(v)}(I')$ the weight of the independent set consisting of the paths of I' and path p and $x_v(I') = (1-\gamma)x_{p(v)}(I')$.
- For each independent set I' of I_v^{27} and I_v^{38} and each path p of M_v^6, we set to $x_v(I) = \frac{\gamma}{L-|M^1|} x_{p(v)}(I')$ the weight of the independent set consisting of the paths of I' and path p and $x_v(I') = (1-\gamma)x_{p(v)}(I')$.
- Let
$$k_1 = \sum_{I \in I^0 \cup I^7 \cup I^8 \cup I^{78}} x_{p(v)}(I)$$
and
$$k_2 = 2n(L - |M_v^1|) + \frac{D(L-|M_v^1|)^2}{L}.$$
- For each independent set I' of $I_v^0 \cup I_v^7 \cup I_v^8 \cup I_v^{78}$ and each path p of M_v^5 and M_v^6, we set to $x_v(I) = \frac{n}{\max\{k_1,k_2\}} x_{p(v)}(I')$ the weight of the independent set consisting of the paths of I' and path p.

- For each independent set I' of $I_v^0 \cup I_v^7 \cup I_v^8 \cup I_v^{78}$ and each pair of paths p, q of M_v^5 and M_v^6 respectively, we set to $x_v(I) = \frac{D}{\max\{k_1,k_2\}L} x_{p(v)}(I')$ the weight of the independent set consisting of the paths of I' and paths p and q.
- For each independent set I of $I_v^0 \cup I_v^7 \cup I_v^8 \cup I_v^{78}$ we set $x_v(I) = \max\{0, 1 - \frac{k_2}{k_1}\} x_{p(v)}(I)$.
- For each path p of M_v^5 and M_v^6, we set to $x_v(I) = n \max\{0, 1 - \frac{k_1}{k_2}\}$ the weight of the independent set consisting only of p.
- For each pair of paths p, q of M_v^5 and M_v^6 respectively, we set to $x_v(I) = \frac{D}{L} \max\{0, 1 - \frac{k_1}{k_2}\}$ the weight of the independent set consisting only of p and q.
- For all other independent sets I (i.e., independent sets that contain at least one path touching only nodes with label greater than v's) we set $x_v(I) = 0$.

Now, we have to set values to the parameters α, β, γ, n so that Fract-Color is correct (i.e., it matches the input-output specification given in the previous section). The next lemma gives sufficient conditions for this purpose.

Lemma 1 (Correctness conditions). *If*

$$0 \leq \alpha, \beta, \gamma \leq 1 \tag{1}$$

$$n \geq 0 \tag{2}$$

$$\alpha L(1-D) + \beta |M_v^1| D + \gamma (L - 2|M_v^1|) D = D(L - |M_v^1|) \tag{3}$$

$$-\beta |M_v^1|^2 D + nL(L - |M_v^1|) = (L - |M_v^1|)((1-D)L - D|M_v^1|) \tag{4}$$

$$2n(L - |M_v^1|) \leq \frac{D^2 - 4D + 4}{3D} L - (4 - 2D)|M_v^1| + \frac{3D|M_v^1|^2}{L} \tag{5}$$

then Fract-Color *is correct.*

Proof. It can be easily verified that conditions (1) and (2) and the fact that the weight assignment $x_{p(v)}$ is non-negative guarantee that x_v is non-negative. Also, by the definition of Fract-Color, x_v satisfies Property 3.

By making calculations, it can be verified that

$$\sum_{I \in \mathcal{I}} x_v(I) = \sum_{I \in \mathcal{I}} x_{p(v)}(I) + \max\{0, k_2 - k_1\}.$$

If $k_1 \geq k_2$, x_v satisfies Property 4 since $x_{p(v)}$ satisfies Property 4. If $k_1 < k_2$, condition (5) and the fact that $x_{p(v)}$ satisfies Property 4 guarantee that x_v satisfy Property 4 as well.

Now, in order to prove Properties 1 and 2 at node v, we have to prove that for any edge e between a node u with label at most the label of v and a child of u, for any path p traversing e, it is $\sum_{I \in I \ : p \in I} x_v(I) = 1 - D$ and for any set of paths p, q traversing e in opposite directions, it is $\sum_{I \in I \ : p,q \in I} x_v(I) = D/L$. We have to distinguish between two cases for e: (i) e is an edge between a node u with label smaller than v's and a child of u and (ii) e is an edge between v and a child of v.

First, we give the proof for case (i). Given an independent set I of I_v, we denote by $J_v(I)$ the set of independent sets that contain all paths of I, (possibly) paths of M_v^5 and M_v^6, and no other paths. Since x_v satisfies Property 3 at node v, it is $x_v(I) = 0$ for any independent set I not belonging to $\bigcup_{I \in I} J_v(I)$. By carefully reading the procedure FRACT-COLOR, we can see that the weight of I under $x_{p(v)}$ equals the sum of weights of the independent sets in $J_v(I)$ under x_v. Also, since $x_{p(v)}$ satisfies Property 3 at $p(v)$, it is $x_{p(v)} = 0$ for all independent sets not belonging to I_v. Furthermore, $x_{p(v)}$ satisfies Properties 1 and 2 at $p(v)$. We obtain that

$$\begin{aligned}\sum_{I \in I \ :p \in I} x_v(I) &= \sum_{I \in I \ \cap I \ :p \in I} \sum_{I' \in J \ (I)} x_v(I') \\ &= \sum_{I \in I \ \cap I \ :p \in I} x_{p(v)}(I) \\ &= \sum_{I \in I \ :p \in I} x_{p(v)}(I) = 1 - D\end{aligned}$$

and

$$\begin{aligned}\sum_{I \in I \ :p,q \in I} x_v(I) &= \sum_{I \in I \ \cap I \ :p,q \in I} \sum_{I' \in J \ (I)} x_v(I') \\ &= \sum_{I \in I \ \cap I \ :p,q \in I} x_{p(v)}(I) \\ &= \sum_{I \in I \ :p,q \in I} x_{p(v)}(I) = D/L.\end{aligned}$$

Proving case (ii) is lengthy. We have to prove it for $e \in \{(v, l(v)), (v, r(v))\}$, and, in each case, we have to dinstinguish between subcases for paths p and q. We will just show that for edge $e = (v, l(v))$ for a path $p \in M_v^5$ and a path $q \in M_v^1$, it is $\sum_{I \in I \ :p,q \in I} x_v(I) = D/L$.

$$\begin{aligned}\sum_{I \in I \ :p,q \in I} x_v(I) &= \sum_{\substack{I \in I^1 \,:\, q \in I \\ I' \in J \ (I) \,:\, p \in I'}} x_v(I') + \sum_{\substack{I \in I^{14} \,:\, q \in I \\ I' \in J \ (I) \,:\, p \in I'}} x_v(I') \\ &\quad + \sum_{\substack{I \in I^{18} \,:\, q \in I \\ I' \in J \ (I) \,:\, p \in I'}} x_v(I') \\ &= \frac{\alpha}{L - |M_v^1|} \sum_{I \in I^1 : q \in I} x_{p(v)}(I) + \frac{\beta}{L - |M_v^1|} \sum_{I \in I^{14} : q \in I} x_{p(v)}(I) \\ &\quad + \frac{\gamma}{L - |M_v^1|} \sum_{I \in I^{18} : q \in I} x_{p(v)}(I) \\ &= \frac{\alpha L(1 - D) + \beta D |M_v^1| + \gamma D (L - 2|M_v^1|)}{L(L - |M_v^1|)} \\ &= D/L\end{aligned}$$

The last equality follows by condition (3). Conditions (3) and (4) are enough to prove all possible subcases. □

In order to prove that FRACT-COLOR is correct, we will compute values for α, β, γ, n such that the correctness conditions (1)–(5) of Lemma 1 are satisfied. We distinguish between two cases according to the size of M_v^1.

CASE I. $|M_v^1| \geq \frac{1-D}{D} L$. In this case, we have the following settings

$$\begin{aligned} \alpha &= 1 \\ \beta &= \frac{(L - |M_v^1|)(D|M_v^1| - (1-D)L)}{D|M_v^1|^2} \\ \gamma &= \frac{L(1-D)}{|M_v^1|D} \\ n &= 0 \end{aligned}$$

Clearly, α satisfies condition (1) and n satisfies condition (2). By simple calculations, we obtain that (3), (4), and (5) are also satisfied. Since $|M_v^1| \geq \frac{1-D}{D} L$, it is $\beta \geq 0$. Also,

$$\begin{aligned} \beta - 1 &= \frac{(L - |M_v^1|)(D|M_v^1| - (1-D)L) - D|M_v^1|^2}{D|M_v^1|^2} \\ &= \frac{-2D|M_v^1|^2 + L|M_v^1| - (1-D)L^2}{D|M_v^1|^2} \end{aligned}$$

The expression in the last line is always non-positive provided that $D \in \left[2/3, \frac{2+\sqrt{2}}{4}\right]$. Thus, β satisfies condition (1). Also, since $\frac{1-D}{D} L \leq |M_v^1| \leq L/2$, γ satisfies (1).

CASE II. $|M_v^1| \leq \frac{1-D}{D} L$. In this case, we have the following settings

$$\begin{aligned} \alpha &= \frac{D|M_v^1|}{(1-D)L} \\ \beta &= 0 \\ \gamma &= 1 \\ n &= 1 - D - \frac{D|M_v^1|}{L} \end{aligned}$$

Clearly, β and γ satisfy condition (1). By making simple calculations, we obtain that (3), (4), and (5) are satisfied. Since $|M_v^1| \leq \frac{1-D}{D} L$, n satisfies condition (2). Also, since $0 \leq |M_v^1| \leq \frac{1-D}{D} L$, α satisfies condition (1).

6 Extensions to the Path Coloring Problem

In this section we outline our path coloring algorithm. Similar ideas are used in [1] to prove a weaker result for general (i.e., non-locally-symmetric) sets of paths

on binary trees. Let T be a binary tree and $\mathcal{P}$ a normal locally-symmetric set of paths on T. Our algorithm uses a rational parameter $D \in [2/3, \frac{2+\sqrt{2}}{4})$. We start with a few definitions.

Definition 5. *Let v be a node of T different from the root and $\mathcal{P}$ a normal locally-symmetric set of paths. A probability distribution $\mathcal{Q}$ over all proper colorings of paths of $\mathcal{P}$ traversing the edge e between v and its parent with $(2-D)L$ colors is weakly uniform if for any two paths $p, q \in \mathcal{P}$ that traverse e in opposite directions, the probability that p and q have the same color is D/L.*

Let v be a non-leaf node of T and let $\mathcal{C}$ be a coloring of paths of $\mathcal{P}$ that traverse the edge e between v and its parent. Let χ be a color used by $\mathcal{C}$. χ is called single color if it is used in only one path traversing e and double color if it is used in two paths traversing e in opposite directions. We denote by A_v^i the set of single colors assigned to paths in M_v^i, and by A_v^{ij} the set of double colors assigned to paths in M_v^i and M_v^j.

Definition 6. *Let v be a non-leaf node of T and $\mathcal{P}$ a normal locally-symmetric set of paths. A weakly uniform probability distribution $\mathcal{Q}$ over all proper colorings of paths of $\mathcal{P}$ traversing the edge e between v and its parent with $(2-D)L$ colors is strongly uniform if*

$$|A_v^{ij}| = \frac{D|M_v^i||M_v^j|}{L}$$

for any pair i, j such that paths of M_v^i traverse e in opposite direction to paths of M_v^j.

First, the algorithm performs the following initialization step to produce a random coloring of the paths touching r with $(2-D)L$ colors according to the weakly uniform probability distribution. It colors the paths originating from the root r. It selects uniformly at random without replacement DL colors of the L colors used to color the paths originating from r and assigns them to DL paths which are selected uniformly at random without replacement from the L paths terminating at r. It also assigns $(1-D)L$ new colors to the paths terminating at r which have not been colored.

Then, for $i = 1, \ldots, n-1$, the algorithm executes the procedures RECOLOR and COLOR at node v with label i.

The procedure RECOLOR at a node v takes as input a random coloring $\mathcal{C}_v$ of the paths traversing the edge between v and its parent with $(2-D)L$ colors according to the weakly uniform probability distribution. RECOLOR stops if v is a leaf. Otherwise, it produces a new random coloring $\mathcal{C}'_v$ of the paths traversing the edge between v and its parent with $(2-D)L$ colors according to the strongly uniform probability distribution.

The procedure COLOR at a node v takes as input a random coloring $\mathcal{C}'_v$ of the paths traversing the edge between v and its parent with $(2-D)L$ colors according to the strongly uniform probability distribution. COLOR stops if v is a leaf. Otherwise, it produces a random coloring $\mathcal{C}''_v$ of the paths touching node v in such a way that:

- The restriction $\mathcal{C}_{l(v)}$ of $\mathcal{C}''_v$ on the paths traversing the edge between v and $l(v)$ is a random coloring with $(2-D)L$ colors according to the weakly uniform probability distribution.
- The restriction $\mathcal{C}_{r(v)}$ of $\mathcal{C}''_v$ on the paths traversing the edge between v and $r(v)$ is a random coloring with $(2-D)L$ colors according to the weakly uniform probability distribution.
- The coloring $\mathcal{C}''_v$ uses at most $\frac{4D^2-4D+4}{3D}$ colors.

Finally, the algorithm uses a deterministic algorithm to color the set of paths whose color has been changed by RECOLOR.

In the following we briefly discuss the main ideas used for the precise definition of procedures RECOLOR and COLOR. We remark that the procedures RECOLOR and COLOR can be implemented to run in time polynomial in L, thus, our algorithm runs in time polynomial in L and the number of nodes of the tree.

Let v be a non-leaf node of T. Observe that if $\mathcal{C}_v$ is a random coloring of the paths traversing the edge between v and its parent, then the numbers $|A_v^{ij}|$ are random variables with expectation

$$\mathcal{E}[|A_v^{ij}|] = \frac{|M_v^i||M_v^j|D}{L}.$$

In [1], it is shown that $|A_v^{ij}|$'s follow a hypergeometrical-like distribution and are sharply concentrated around their expectations. The procedure RECOLOR uses this property and, with high probability, by recoloring a small number of paths traversing the edge between v and its parent, it produces a random coloring $\mathcal{C}'_v$ which follows the strongly uniform probability distribution. This is done in such a way that after all executions of RECOLOR the load of the paths of $\mathcal{P}$ whose color has been changed at least once is $o(L)$ with high probability.

The procedure COLOR mimics FRACT-COLOR in some sense. During its execution at a non-leaf node v such that P_v belongs to Scenario I (the case of Scenario II is similar to Scenario II of FRACT-COLOR) it colors the paths of M_v^5 and M_v^6 using colors of A_v^i and A_v^{ij} in such a way that the probability that two paths traversing one of the edges between v and a child of v in opposite directions are assigned the same color equals D/L (recall that this equals the sum of the weights x_v produced by FRACT-COLOR of the independent sets containing both paths). The analysis is much similar to the analysis of FRACT-COLOR. By working on the details of the analysis, we conclude to similar correctness conditions with those of FRACT-COLOR.

We now state our result for the path coloring of locally-symmetric sets of paths.

Theorem 2. *There exists a randomized polynomial-time algorithm which, for any constant $\delta < 1/3$, colors any locally-symmetric set of paths of load L on a binary tree of depth at most $L^\delta/8$, using at most $1.367L + o(L)$ colors, with probability at least $1 - \exp\left(-\Omega(L^\delta)\right)$.*

The restriction in the depth of the tree is necessary so that the upper bound on the number of colors is guaranteed with high probability. The number of nodes of the tree may be exponential in L (i.e., up to $2^{L/8}$).

References

1. V. Auletta, I. Caragiannis, C. Kaklamanis, and P. Persiano. Randomized Path Coloring on Binary Trees. *Theoretical Computer Science,* Vol. 289(1), 2002, pp. 355-399.
2. I. Caragiannis, A. Ferreira, C. Kaklamanis, S. Perennes, and H. Rivano. Fractional Path Coloring with Applications to WDM Networks. In *Proc. of the 28th International Colloquium on Automata, Languages, and Programming (ICALP '01)*, LNCS 2076, Springer, 2001, pp. 732–743.
3. I. Caragiannis, C. Kaklamanis, and P. Persiano. Symmetric Communication in All–Optical Tree Networks. *Parallel Processing Letters*, Vol. 10(4), 2000, pp. 305–313.
4. S. Corteel, D. Gardy, D. Barth, A. Denise, and M. Valencia-Pabon. On the Complexity of Routing Permutations on Trees by Arc-Disjoint Paths. In *Proc. of the 4th LATIN: Theoretical Informatics*, Springer, LNCS 1776, 2000, pp. 308-317.
5. T. Erlebach and K. Jansen. The Complexity of Path Coloring and Call Scheduling. *Theoretical Computer Science*, Vol. 255(1–2), 2000, pp. 33–50.
6. T. Erlebach, K. Jansen, C. Kaklamanis, M. Mihail, and P. Persiano. Optimal Wavelength Routing in Directed Fiber Trees. *Theoretical Computer Science*, Vol. 221(1–2), 1999, pp. 119–137.
7. V. Kumar. Approximating Circular Arc Colouring and Bandwidth Allocation in All–Optical Ring Networks. In *Proc. of the 1st International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX '98)*, LNCS 1444, Springer, 1998, pp. 147-158.
8. V. Kumar and E. Schwabe. Improved Access to Optical Bandwidth in Trees. In *Proc. of the 8th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA '97)*, 1997, pp. 437–444.
9. P. Raghavan, E. Upfal. Efficient Routing in All-Optical Networks. In *Proc. of the 26th Annual Symposium on Theory of Computing (STOC '94)*, 1994, pp. 133–143.
10. R. Ramaswami, K. Sivarajan. Optical Networks: A Practical Perspective. *Morgan Kauffman Publishers*, 1998.

A $\frac{5}{4}$-Approximation Algorithm for Scheduling Identical Malleable Tasks*

Thomas Decker, Thomas Lücking, and Burkhard Monien

University of Paderborn
Fürstenallee 11
33102 Paderborn, Germany
Fax: +49 (5251) 60 6697
{decker,luck,bm}@uni-paderborn.de

Abstract. We consider the problem of finding a schedule for n identical malleable tasks on p identical processors with minimal completion time. This problem arises while using the branch & bound or the divide & conquer strategy to solve a problem on a parallel system. If nothing is known about the sub-problems, then they are assumed to be identical. We assume that the execution time decreases with the number of processors while the computational work increases. We give an algorithm with running time exponential in p which computes an optimal schedule. In order to approximate an optimal schedule, we use the concept of phase-by-phase schedules. Here schedules consist of phases in which every job uses the same number of processors. We prove that one can approximate an optimal schedule up to a factor $\frac{5}{4}$ using constant time, and we show that this is optimal. Furthermore, we give an ε-approximation algorithm if the speed-up is optimal up to a constant factor.

1 Introduction

Motivation-Framework. The multiprocessor scheduling problem for identical processors is well-known and has been studied extensively in many different variations. If the number of processors on which a specific job has to be executed is part of the input, then the tasks are called non-malleable. Otherwise they are called malleable. Likewise, if the jobs may be interrupted while being executed, then the resulting schedule is called preemptive, otherwise it is called non-preemptive. Furthermore, the specification depends on precedence constraints between jobs, and on the objective. Veltman et al. [22] give an overview of a multitude of works on various specifications.

In this paper we consider the problem of finding a non-preemptive schedule for n identical jobs on p identical processors with minimal total completion time, the so-called makespan. Since the jobs are identical, the time function is equal

* Partly supported by the DFG-Sonderforschungsbereich 376 Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen, and by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 95–108, 2004.
© Springer-Verlag Berlin Heidelberg 2004

for all jobs, that is, the execution time on any certain number of processors is the same for all jobs.

Using the branch & bound or the divide & conquer strategy to solve a problem, the problem is split into smaller subproblems which have to be solved, that is, tasks which have to be executed. In many cases the parallelism given by the branch & bound tree or by the divide & conquer tree is sufficient to yield a good speed-up. But in many other cases this is not true and we have to parallelize the computations done at the tree nodes. All these computations are of the same kind and so we may assume that all these tasks are identical. In this case the scheduling problem we consider applies. Our motivation for doing research on this problem is that this scheduling problem arises isolating real roots in parallel using the Descartes method [8]. As far as we know there is no publication on this scheduling problem. We assume that the same properties for the execution time as in [3] hold. This implies that the execution of the jobs achieves some speed-up, but no super-linear speed-up. Up to now, it is not clear whether the corresponding decision problem is $\mathcal{NP}$-hard or not.

Related work. There exist many results on the complexity of various scheduling problems [5]. Du and Leung [10] showed that both the non-malleable and the malleable scheduling problem are $\mathcal{NP}$-hard, so researchers are interested in approximation algorithms. If the tasks are non-malleable, then the scheduling problem is a special case of the resource constraint scheduling problem. Hence the optimal schedule can be approximated up to factor 2 using list planning [12].

Krishnamurti and Ma [17] were the first who did research on the scheduling problem with malleable tasks. Belkhale and Banerjee [3] introduced an algorithm with approximation factor $2/(1 - 1/p)$, assuming that execution time decreases with the number of processors while the computational work increases. Turek et al. [21] improved this result, using no assumptions, and showed an approximation factor 2. The latest result is from Mounié et al. [18]. Using the same assumptions for the execution time as Belkhale and Banerjee [3], they proved that an optimal schedule for malleable tasks can be approximated up to the factor $\sqrt{3}$. Blazewicz et al. [4] improved this result to 2, and a proof for the factor $\frac{3}{2}$ has been submitted for publication [19].

The best known approximation algorithm for schedules where jobs are only allowed to be executed on processors with successive indices is from Steinberg [20]. He showed an approximation factor 2. Note that this scheduling problem is closely related to the orthogonal packing problem of rectangles first investigated by Baker et al. [1]. See [11, 2, 16] for a typology of cutting and packing problems and results on approximation algorithms.

Jansen and Porkolab [14] invented the first polynomial approximation algorithm for a constant number of processors using linear programming. Note that this problem is related to orthogonal strip packing of rectangles [6, 15]. Furthermore, Jansen [13] proposed an asymptotic fully polynomial time approximation scheme if p is part of the input.

Results. In Section 2 we formally define the scheduling problem and introduce an algorithm which computes a schedule with minimal makespan. We show that its running time is exponential in the number p of processors. If p is constant, then this yields an algorithm polynomial in the number n of jobs. However, the algorithm is not suitable for practical purposes. In order to approximate an optimal schedule, we introduce phase-by-phase schedules in Section 3. Here schedules have a simple structure. They consist of phases in which each job uses the same number of processors. A new phase can not be started until the last phase has finished. We illustrate with help of an example that the quotient of the makespan of an optimal phase-by-phase schedule and the makespan of an optimal schedule can be $\frac{5}{4}$. Furthermore, we give a constant time algorithm which only uses certain phase-by-phase schedules providing an approximation factor $\frac{5}{4}$. In order to prove this approximation factor, we prove two technical lemmas in Section 4, providing good lower bounds on the makespan of an optimal schedule as well as restricting the set of schedules which have to be considered to dominating schedules. In Sections 5 we use these lemmas to prove the correctness of the algorithm for the case $p < n \leq \lfloor \frac{3}{2}p \rfloor$ by case analysis. Due to lack of space the proofs of the other cases are omitted here. They can be done in a similar way and can be found in [9]. Finally, we give an ε-approximation algorithm in the case that the speed-up is optimal up to a constant factor in Section 6.

2 Optimal Schedules

For the malleable scheduling problem the input consists of n, p and t: n is the number of identical jobs which can be executed on different number of processors in parallel. p is the number of identical processors on which the jobs are to be scheduled. t is the time function given by $t : \{1, \ldots, p\} \longrightarrow \mathbb{R}^+$. Here $t(i)$ is the running time needed to compute a job on i processors. For $1 \leq i \leq j \leq p$ the time function t must have the following properties:

- **monotony**: $t(j) \leq t(i)$
- **speed-up property**: $i \cdot t(i) \leq j \cdot t(j)$

These properties imply that the execution of the jobs may achieve some speed-up, but no super-linear speed-up.

A **schedule** S assigns a set σ_i of processors and a starting time τ_i to every job i such that all jobs are executed and every processor executes at most one job at any time. We call (σ_i, τ_i) the **plan** according to job i. The objective is to minimize the **makespan** defined by

$$\max\{\tau + t(|\sigma|) \mid (\sigma, \tau) \in S\}.$$

A schedule with minimal makespan is called **optimal schedule**.

Our next goal is to find an algorithm to compute an optimal schedule. Some of these results are also in [7]. In order to proof the correctness of our algorithm, we first show that there always exists an optimal schedule with certain properties.

Definition 1.

1. *The* **load vector** $l(S) = (l_1, \ldots, l_p)$ *of a schedule* S *is defined by*

$$l_i = \max\{\tau + t(|\sigma|) \mid (\sigma, \tau) \in S \wedge i \in \sigma\},\ 1 \leq i \leq p.$$

2. *The* **sorted load vector** $\lambda(S) = (\lambda_1, \ldots, \lambda_p)$ *is given by* $\lambda_i = l_{\pi(i)}$, $1 \leq i \leq p$, *where* π *is a permutation with* $l_{\pi(i)} \leq l_{\pi(j)}$ *for all* $1 \leq i < j \leq p$. *Note that* λ_p *is the makespan of* S.

We will show that for each schedule there exists another schedule with the same makespan and certain properties. One of this properties is that each job in a schedule starts either at time $\tau = 0$ or directly subsequent to another job. We now define this property formally.

Definition 2. *A schedule* S *is* **packed** *if for all plans* $(\sigma, \tau) \in S$ *either* $\tau = 0$ *holds, or there exists another pair* $(\sigma', \tau') \in S$ *with* $\sigma \cap \sigma' \neq \emptyset$ *and* $\tau' + t(|\sigma'|) = \tau$.

Definition 3. *Let* S *be a schedule for* n *jobs. Then* S *contains a schedule* S' *if* $S' \subseteq S$. *The schedules* $S_1, \ldots, S_{n-1}$ *are called* **intermediary schedules of** S *if* $S_1 \subseteq \ldots \subseteq S_{n-1} \subseteq S$ *and* $|S_i| = i$ *for all* $1 \leq i < n$.

In general we can compute a schedule for $i > 1$ jobs by using a schedule for $i - 1$ jobs and assigning a set of free processors to the ith job. We use this fact to give an algorithm. Every step leads to a set of intermediary schedules.

Lemma 1. *For each optimal schedule* S *for* n *jobs there exists a schedule* S_n *and intermediary schedules* $S_1, \ldots, S_{n-1}$ *of* S_n *such that for* $1 \leq i \leq n$ *we have*

1. S *and* S_n *have the same makespan.*
2. S_i *is packed, and for the sorted load vector* $(\lambda_1^{(i)}, \ldots, \lambda_p^{(i)})$ *of* S_i *the inequality* $\lambda_p^{(i)} - \lambda_1^{(i)} \leq t(1)$ *holds. No job in* S_i *is finished later than in* S.

Proof. Let $1 \leq i \leq n$, and let S_{i-1} be a packed intermediary schedule. Let j be a processor with $l_j^{i-1} = \lambda_1^{i-1}$, i.e. j has minimal load among all processors in S_{i-1}. Choose a plan $(\sigma, \tau) \in S$ with $j \in \sigma$ and minimal starting time which has not been considered yet. Denote $S' = S_{i-1} \cup \{(\sigma, \tau)\}$.

If S' is packed then we set $S_i := S'$, and we are done. Otherwise let $\tau^\star$ be the earliest time at which all processors in σ in S_{i-1} are ready. Then $S^\star := S_{i-1} \cup \{(\sigma, \tau^\star)\}$ is packed, and $S^\star \cup (S \setminus S')$ is an optimal schedule. Hence $S_i := S^\star$ is an intermediary schedule.

If the makespan of S' is larger than $\lambda_1^{i-1} + t(1)$, then remove (σ, τ) from the schedule and add the plan $(\{j\}, l_j^{i-1})$. Proceed in the same way if there exists no $(\sigma, \tau) \in S \setminus S_{i-1}$ with $j \in \sigma$ and the makespan of S is larger than $\lambda_1^{i-1} + t(1)$. If no such (σ, τ) exists and the makespan of S is at most $\lambda_1^{i-1} + t(1)$ then we may proceed with any plan from $S \setminus S_{i-1}$ with minimal starting time. □

We get optimal schedules by computing intermediary schedules of optimal schedules iteratively. Due to Lemma 1 we can restrict our search to packed schedules with sorted load vector $(\lambda_1, \ldots, \lambda_p)$ with $0 \leq \lambda_p - \lambda_1 \leq t(1)$.

In the following we will present optimal algorithms. We do so for time functions $t : \mathbb{N} \to \mathbb{R}^+$ and $t : \mathbb{N} \to \mathbb{N}$. In both cases we assume that manipulating numbers can be done in constant time. Applying the Turing machine model would result in an additional factor $\mathcal{O}(\log(\text{makespan})) = \mathcal{O}(\log(n)+\log(t(1)))$ if $\mathbb{N}$ is the range of the time function. Note that the jobs are identical and therefore the number n of jobs contributes only $\log(n)$ bits to the input size.

Lemma 2. *If a schedule S is packed, then each entry of the load vector $\lambda(S) = (\lambda_1, \ldots, \lambda_p)$ can be written as a linear combination*

$$\lambda_i \in \{a_1 t(1) + \cdots + a_p t(p) \mid a_1, \ldots, a_p \in \{0, \ldots, n\}\}.$$

Theorem 1.

1. *If the range of the time function t is $\mathbb{R}^+$, then there exists an algorithm which computes an optimal schedule using $\mathcal{O}(n(n+1)^{p^2}2^p)$ time.*
2. *If the range of the time function t is $\mathbb{N}$, then there exist algorithms which compute an optimal schedule using $\mathcal{O}(\log(n) \cdot |V|^3)$ or $\mathcal{O}(n \cdot |V| \cdot 2^p)$ time, where $|V| = t(1)^p$.*

Proof. We can view the scheduling problem as a constrained shortest path problem in a directed graph. For a sorted load vector $\lambda(S) = (\lambda_1, \ldots, \lambda_p)$ of a schedule S we denote with $\Lambda(S) = (\lambda_i - \lambda_1)_{i=1,\ldots,p}$ the pattern of S. In the directed graph every possible pattern corresponds to a node. Let V be the set of nodes. A directed edge from node i to node j exists if and only if we can get a schedule with the pattern corresponding to node j by adding a plan to a schedule with the pattern corresponding to node i. We assign to each edge (i, j) the augmentation of the makespan gained by adding the corresponding plan. Now we can compute the minimal makespan by searching a path with n edges having minimal weight.

1. If the range of the time function t is $\mathbb{R}^+$, then Lemma 2 yields $|V| \leq (n+1)^{p^2}$. Furthermore we know that the graph has an out-degree at most 2^p. We use dynamic programming to compute all paths with i edges for $1 \leq i \leq n$. So we need at most $\mathcal{O}(n(n+1)^{p^2}2^p)$ time.
2. If the range of the time function t is $\mathbb{N}$, then we have $|V| = t(1)^p$. We can compute a minimal weight path with n edges in $\mathcal{O}(\log(n) \cdot |V|^3)$ time using the doubling-technique. The time complexity of the second claim can be proved in the same way as in part (1) using the fact that we only have to consider $|V| = t(1)^p$ nodes for $1 \leq i \leq n$. □

We have proved that it is possible to compute an optimal schedule using time polynomial in n if the number of processors is constant. However, the running time depends exponentially on the number of processors.

3 Phase-by-Phase Schedules (PPS)

We will use a **phase-by-phase schedule (PPS)** to approximate an optimal schedule. Here a schedule consists of phases in which each job uses the same

number of processors. A new phase can not be started until the last phase has finished. Hence we do not have to store plans for all jobs but may write a phase-by-phase schedule with k phases as $PPS = (i_1, \ldots, i_k)$ where i_j denotes the number of processors used in phase j. The makespan is

$$\text{makespan}(PPS) = \sum_{j=1}^{k} t(i_j).$$

Decker and Krandick [8] introduced an algorithm which computes an optimal phase-by-phase schedule using $\mathcal{O}(n^2)$ time. The running time of this algorithm can be improved to $\mathcal{O}(n \cdot \min\{n, p\})$. We now show that there exists an algorithm which computes an optimal phase-by-phase schedule using $\mathcal{O}(p^3 + \log(n))$ time.

Theorem 2. *There exists an algorithm which computes an optimal phase-by-phase schedule using $\mathcal{O}(p^3 + \log(n))$ time.*

Proof. Let x_i be the number of phases using i processors. Due to the speed-up property $x_i \leq i - 1$ holds for $i \geq 2$. Furthermore at most $\lfloor \frac{p}{i} \rfloor$ jobs are executed during such a phase. Therefore at most

$$\sum_{2 \leq i \leq p} x_i \left\lfloor \frac{p}{i} \right\rfloor \leq \sum_{2 \leq i \leq p} (i-1) \left\lfloor \frac{p}{i} \right\rfloor \leq (p-1)p$$

jobs are executed in phases using more than one processor.

The algorithm works as follows. If $n > (p-1)p$, then compute the minimal r such that $n - r \cdot p \leq (p-1)p$. The schedule starts with r sequential phases. The improved algorithm from [8] is used to compute the schedule for the remaining $n - r \cdot p$ jobs. □

Decker [7] showed that the makespan of an optimal phase-by-phase schedule is at most twice as large as the makespan of an optimal schedule. The following example illustrates that computing an optimal phase-by-phase schedule can not lead to an approximation factor lower than $\frac{5}{4}$.

Example 1. Let $p = 5$ and $n = 3$. Furthermore let $t(1) = 1$, $t(2) = \frac{1}{2}$ and $t(i) = \frac{1}{3}$ for all $3 \leq i \leq 5$. The optimal phase-by-phase schedule is $(2, 5)$ with makespan $t(2) + t(5) = \frac{5}{6}$. However, $\sigma = \{(\{1,2,3\}, 0), (\{1,2,3\}, \frac{1}{3}), (\{4,5\}, 0)\}$ is an optimal schedule with makespan $2t(3) = \frac{2}{3}$. So the approximation factor via phase-by-phase schedules is $\frac{5}{4}$. Note that this example can be extended to $p = 3j + 2$ and $n = 2j + 1$ with $j \in \mathbb{N}$.

Our next goal is to find an algorithm with approximation factor $\frac{5}{4}$ using constant time. The idea of our algorithm is as follows: we compute a solution only depending on the relation between p and n. We do not know anything about the time function (up to the monotony and the speed-up property), and we will consider a phase-by-phase schedule. In the following we show that this algorithm provides an approximation factor of $\frac{5}{4}$. For the sake of readability we only return the makespan of the phase-by-phase schedule which we choose. Note that the phase-by-phase schedule is returned, too. The algorithm is as follows:

Algorithm 1 *Phase-by-Phase Schedule (PPS)*
Input: *number of jobs n, number of processors p, time function t*
Output: *phase-by-phase schedule*

```
(1)   begin
(2)   if n > p then
(3)       if p < n ≤ ⌊3/2 p⌋ then
(4)           a = ⌊ p / (n − p) ⌋
(5)           return t(1) + t(a)
(6)       if ⌊3/2 p⌋ < n ≤ 2p then
(7)           a = ⌊ p / (n − ⌊3/2 p⌋) ⌋
(8)           return min{t(1) + t(2) + t(a), 2t(1)}
(9)       if 2p < n ≤ ⌊5/2 p⌋ then
(10)          a = ⌊ p / (n − 2p) ⌋
(11)          return 2t(1) + t(a)
(12)      if ⌊5/2 p⌋ < n ≤ 3p then
(13)          a = ⌊ p / (n − ⌊5/2 p⌋) ⌋
(14)          return min{2t(1) + t(2) + t(a), 3t(1)}
(15)      else
(16)          a = ⌊ p / (n − ⌊−⌋ · p) ⌋
(17)          return ⌊n/p⌋ · t(1) + t(a)
(18)  else
(19)      if ⌊p/2⌋ < n ≤ p then
(20)          a = ⌊ p / (n − ⌊p/2⌋) ⌋
(21)          if 2⌊p/3⌋ < n ≤ p then
(22)              return min{t(1), t(2) + t(a)}
(23)          if ⌊p/3⌋ + ⌊p/4⌋ < n ≤ 2⌊p/3⌋ then
(24)              return min{t(1), t(2) + t(a), 2t(3)}
(25)          else
(26)              return min{t(1), t(2) + t(a), t(3) + t(4)}
(27)      if ⌊p/3⌋ < n ≤ ⌊p/2⌋ then
(28)          a = ⌊ p / (n − ⌊p/3⌋) ⌋
(29)          if 2/5 p ≤ n ≤ ⌊p/2⌋ then
(30)              return min{t(2), t(3) + t(a)}
(31)          if 2⌊p/5⌋ < n < 2/5 p then
(32)              if p ≤ 17 then
(33)                  return min{t(2), t(3) + t(a), t(4) + t(6)}
(34)              else
(35)                  return min{t(2), t(3) + t(a), t(4) + t(5)}
(36)          else
(37)              return min{t(2), t(3) + t(a), 2t(5)}
(38)      if ⌊p/4⌋ < n ≤ ⌊p/3⌋ then
(39)          a = ⌊ p / (n − ⌊p/4⌋) ⌋
(40)          if 2/7 p ≤ n ≤ ⌊p/3⌋ then
(41)              return min{t(3), t(4) + t(a)}
(42)          if 2⌊p/7⌋ < n < 2/7 p then
```

```
(43)                    if rem(p, 7) = 5 and p ≤ 36 then
(44)                        return min{t(3), t(4) + t(a), t(5) + t(9), t(6) + t(7)}
(45)                    if rem(p, 7) = 6 and p ≤ 37 then
(46)                        return min{t(3), t(4) + t(a), t(5) + t(11), t(6) + t(7)}
(47)                    else
(48)                        return min{t(3), t(4) + t(a), t(6) + t(7)}
(49)                else
(50)                    return min{t(3), t(4) + t(a), 2t(7)}
(51)            if ⌊p/k⌋ < n ≤ ⌊p/(k−1)⌋ and k ≥ 5 then
(52)                a = ⌊p/(n − ⌊n/k⌋)⌋
(53)                return min{t(k − 1), t(k) + t(a)}
(54)    end
```

We now show that Algorithm 1 approximates an optimal schedule up to a factor $\frac{5}{4}$ using constant time. Note that applying the Turing machine model would result in a time bound $\mathcal{O}(\log(n) + \log(p) + \log(t(1)))$. The proof for $n \leq p$ is more complex, mostly since we aimed to get the real approximation factor for this interval. Thus the algorithm for $n \leq p$ is rather complex. The approximation factor for the simple algorithm we use for $n > p$ is $\frac{6}{5}$. However, we think that by using a more complex algorithm, we could get a better result. The example shown in Figure 1 demonstrates that for $n > p$ the approximation factor is at least $\frac{8}{7}$.

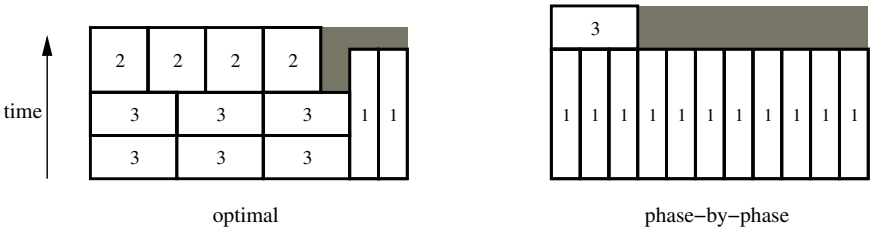


Fig. 1. Optimal schedule and optimal phase-by-phase schedule for $p = 11$, $n = 12$, and $t(1) = 1$, $t(2) = \frac{1}{2}$ and $t(i) = \frac{1}{3}$ for all $3 \leq i \leq 11$. The makespan of the optimal schedules is $t(2) + 2t(3) = \frac{7}{6}$, and the makespan of the optimal phase-by-phase schedule is $t(1) + t(3) = \frac{4}{3}$. So the approximation factor is $\frac{8}{7}$.

Theorem 3. *Algorithm 1 computes a phase-by-phase schedule which is an optimal schedule up to a factor $\frac{5}{4}$ using constant time. If $n > p$ then the approximation factor is $\frac{6}{5}$.*

Proof. For $p < n \leq \lfloor \frac{3}{2}p \rfloor$ the claim follows with Lemma 5. Due to lack of space the proofs for the other cases are omitted here. They can be done in a similar way and can be found in [9].

4 Technical Lemmas

In the following we denote with T_{OPT} the makespan of an optimal schedule. In order to prove Theorem 3 we mainly consider the quotient of the upper bound provided by Algorithm 1 and a lower bound for T_{OPT}. The following lemma will help us to find good lower bounds.

Lemma 3. *Let $1 \leq u_1 \leq u_2 \leq p$, $\frac{p}{u_1} + \frac{p}{u_2} \leq n$, and let the makespan of an optimal schedule consist of at least two addends.*

1. *If $t(u_1)$ is an addend in the makespan of an optimal schedule, then $T_{OPT} \geq t(u_1) + t(u_2)$.*
2. *If $u_2 \leq u_1 + 1$ and no $t(i)$, $i < u_1$, is an addend in the makespan of an optimal schedule, then $T_{OPT} \geq t(u_1) + t(u_2)$.*

Proof.

1. Let $t(u_1) + \sum_{1 \leq i \leq l} t(k_i)$ be the makespan of an optimal schedule. If there exists $k_i \leq u_2$, then we are done. Otherwise the speed-up property yields

$$t(u_1) + \sum_{1 \leq i \leq l} t(k_i) \geq t(u_1) + \sum_{1 \leq i \leq l} \frac{u_2}{k_i} \cdot t(u_2) = t(u_1) + \frac{u_2}{p} \cdot t(u_2) \sum_{1 \leq i \leq l} \frac{p}{k_i}$$
$$\geq t(u_1) + \frac{u_2}{p} \cdot t(u_2) \cdot \left(n - \frac{p}{u_1} \right) \geq t(u_1) + t(u_2).$$

2. Let $\sum_{1 \leq i \leq l} t(k_i)$ be the makespan of an optimal schedule. If $t(u_1)$ is an addend in this makespan, then we are done. Otherwise we have $k_i \geq u_2$ for all $1 \leq i \leq l$. We get

$$\sum_{1 \leq i \leq l} t(k_i) \geq \sum_{1 \leq i \leq l} \frac{u_2}{k_i} \cdot t(u_2) = \frac{u_2}{p} \cdot t(u_2) \cdot \sum_{1 \leq i \leq l} \frac{p}{k_i} \geq \frac{u_2}{p} t(u_2) \cdot n$$
$$\geq \frac{u_2}{p} \cdot t(u_2) \left(\frac{p}{u_1} + \frac{p}{u_2} \right) = t(u_2) + \frac{u_2}{u_1} \cdot t(u_2) \geq t(u_1) + t(u_2).$$

This completes the proof of the lemma. □

We have to consider all makespans of possible schedules. Unfortunately the number of such makespans can become very large. So finding a good lower bound is difficult. The next definition helps us to restrict the number of makespans we have to consider.

Definition 4. *Let S, T be two packed schedules, and let the sums $\sum_{1 \leq l \leq k_1} t(i_l)$ and $\sum_{1 \leq l \leq k_2} t(j_l)$ be the makespans of S and T, respectively. Then S* **dominates** *T if for all valid time functions t, that is, time functions for which the monotony and the speed-up property hold,*

$$\sum_{1 \leq l \leq k_1} t(i_l) < \sum_{1 \leq l \leq k_2} t(j_l).$$

Lemma 4. *Let S be a schedule, $r \in \mathbb{Q}^+$, $n > p \cdot r$, and let $I \subseteq \{m \in \mathbb{N} \mid m \leq p\}$ be the set of number of processors which are allowed to be used. Then*

$$T_{OPT} \geq \min \left\{ \sum_{i \in I} x_i t(i) \,\middle|\, \sum_{i \in I} \frac{x_i}{i} > r \right\}.$$

Proof. Let $1 \leq j \leq p$, and let $y_i(j)$ be the number of jobs in an optimal schedule which use i processors including processor j. Now consider a solution to the problem. If there exists a j with $\sum_{i \in I} \frac{y_{(j)}}{i} > r$, then we are done. Otherwise we have

$$\sum_{1 \leq j \leq p} \sum_{i \in I} \frac{y_i(j)}{i} = \sum_{i \in I} \sum_{1 \leq j \leq p} \frac{y_i(j)}{i} \leq p \cdot r < n,$$

which contradicts the solvability. □

Due to Lemma 4 we only have to consider the makespans of dominating schedules for which $\sum_{i \in I} \frac{x}{i} > r$.
In order to find another way to get a lower bound on T_{OPT}, we use the speed-up property as follows: if each job in the schedule uses at least j processors, then each job needs at least $j \cdot t(j)$ area. This leads to the lower bound

$$T_{OPT} \geq \frac{n}{p} \cdot (j \cdot t(j)).$$

In the following this technique will be helpful.

5 Proof for the Case $p < n \leq \lfloor \frac{3}{2} p \rfloor$

Lemma 5. *Let $p < n \leq \lfloor \frac{3}{2} p \rfloor$. Then Algorithm 1 computes a phase-by-phase schedule which is an optimal schedule up to a factor $\frac{6}{5}$.*

Proof. Let $a = \lfloor \frac{p}{n-p} \rfloor$. Since $n \leq \lfloor \frac{3}{2} p \rfloor$ we have $a \geq 2$. Algorithm 1 returns $t(1) + t(a)$. Note that $t(1)$ is obviously a lower bound on the makespan of an optimal schedule.
If $t(a) \leq \frac{1}{5} t(1)$ we get

$$\frac{t(1) + t(a)}{t(1)} \leq \frac{6}{5}.$$

So let $t(a) > \frac{1}{5} t(1)$.
First case: There exist no more than p jobs each being executed on at most a processors.
Using the speed-up property yields

$$\begin{aligned} T_{OPT} &\geq \frac{1}{p}(p \cdot t(1) + (n-p)(a+1) \cdot t(a+1)) = t(1) + \frac{n-p}{p} \cdot (a+1) \cdot t(a+1) \\ &= t(1) + \frac{n-p}{p} \left(\left\lfloor \frac{p}{n-p} \right\rfloor + 1 \right) t(a+1) > t(1) + t(a+1) \\ &\geq t(1) + \frac{a}{a+1} t(a). \end{aligned}$$

Due to monotony of the time function we get

$$\frac{t(1)+t(a)}{t(1)+\frac{a}{a+1}t(a)} \leq \frac{t(1)+t(a)}{t(1)+\frac{2}{3}t(a)} \leq \frac{t(1)+t(1)}{t(1)+\frac{2}{3}t(1)} \leq \frac{2}{1+\frac{2}{3}} = \frac{6}{5}.$$

Second case: There exist at least $p+1$ jobs each being executed on at most a processors.
We only have to consider dominating schedules for which $t(a)$ is the smallest addend which appears in the makespan. Since $t(a) > \frac{1}{5}t(1)$, we have $T_{OPT} \geq 6t(a) > t(1)+t(a)$ for all makespans of a schedule with more than 5 addends. If we have 5 addends then $T_{OPT} \geq 5t(a) > \frac{4}{5}t(1)+t(a)$, and we get

$$\frac{t(1)+t(a)}{\frac{4}{5}t(1)+t(a)} \leq \frac{1+\frac{1}{5}}{\frac{4}{5}+\frac{1}{5}} = \frac{6}{5}.$$

Hence we only have to consider makespans with at most 4 addends. The dominating schedules are those corresponding to the makespans $t(1)+t(a)$, $t(2)+t(3)+t(a)$, $t(2)+t(5)+2t(a)$ for $a \geq 4$, and $2t(3)+2t(a)$ for $a \geq 3$.
$t(1)+t(a)$:
We immediately get

$$\frac{t(1)+t(a)}{t(1)+t(a)} = 1.$$

$t(2)+t(3)+t(a)$:
For $a \leq 2$ we immediately get the approximation factor 1. For $a = 3$ we have

$$\frac{t(1)+t(3)}{t(2)+2t(3)} \leq \frac{t(1)+t(3)}{\frac{1}{2}t(1)+2t(3)} \leq \frac{1+\frac{1}{3}}{\frac{1}{2}+\frac{2}{3}} \leq \frac{8}{7}.$$

For $a \geq 4$ we get

$$\frac{t(1)+t(a)}{t(2)+t(3)+t(a)} \leq \frac{t(1)+t(a)}{(\frac{1}{2}+\frac{1}{3})t(1)+t(a)} < \frac{t(1)+\frac{1}{5}t(1)}{\frac{5}{6}t(1)+\frac{1}{5}t(1)} \leq \frac{6}{5}\cdot\frac{30}{31} < \frac{6}{5}.$$

$t(2)+t(5)+2t(a)$:
We have

$$\frac{t(1)+t(a)}{t(2)+t(5)+2t(a)} \leq \frac{t(1)+t(a)}{(\frac{1}{2}+\frac{1}{5})t(1)+2t(a)} < \frac{t(1)+\frac{1}{5}t(1)}{\frac{7}{10}t(1)+\frac{2}{5}t(1)} = \frac{6}{5}\cdot\frac{10}{11} < \frac{6}{5}.$$

$2t(3)+2t(a)$:
It follows that

$$\frac{t(1)+t(a)}{2t(3)+2t(a)} \leq \frac{t(1)+t(a)}{\frac{2}{3}t(1)+2t(a)} < \frac{t(1)+\frac{1}{5}t(1)}{\frac{2}{3}t(1)+\frac{2}{5}t(1)} = \frac{6}{5}\cdot\frac{15}{16} < \frac{6}{5}.$$

This completes the proof. □

6 An ε-Approximation Algorithm

In the previous section we proved that there exists a $\mathcal{O}(\log(n))$-time algorithm with approximation factor $\frac{5}{4}$. We now show that if the speed-up is good enough and p is large enough, then an optimal phase-by-phase schedule approximates an optimal schedule up to a factor $1+\varepsilon$. Using $r = \frac{n}{p}$ we denote

$$T_k = \min\left\{ \sum_{i=1}^{k} y_i t(i) \;\middle|\; \sum_{i=1}^{k} \frac{y_i}{i} \geq r \right\}.$$

Note that T_p is a lower bound on the makespan of an optimal schedule (see Section 5). The following lemma is the key to our algorithm.

Lemma 6. *For $1 \leq k \leq p$ we have $T_p \geq T_k - t(k)$.*

Proof. Denote $\sum_{i=k+1}^{p} \frac{y}{i} = \frac{a}{k} + b$ with $a \in \mathbb{N}_0$ and $0 \leq b < \frac{1}{k}$. Furthermore, let

$$z_i = \begin{cases} y_i & \text{if } i < k \\ y_k + a + 1 & \text{if } i = k \end{cases}$$

This implies

$$\sum_{1 \leq k \leq p} \frac{z_i}{i} = \sum_{1 \leq i \leq k-1} \frac{y_i}{i} + \frac{y_k + a + 1}{k} \geq \sum_{i=1}^{k} \frac{y_i}{i} + \frac{k \sum_{k+1 \leq i \leq p} \frac{y}{i}}{k} = \sum_{1 \leq i \leq p} \frac{y_i}{i} \geq r.$$

The speed-up property yields

$$\begin{aligned} T_p &= \sum_{i=1}^{p} y_i t(i) \geq \sum_{i=1}^{k} y_i t(i) + \left(\sum_{i=k+1}^{p} \frac{y_i}{i} \right) k \cdot t(k) = \sum_{i=1}^{k} y_i t(i) + \left(\frac{a}{k} + b \right) k \cdot t(k) \\ &= \sum_{i=1}^{k} z_i t(i) - (a+1)t(k) + (a + bk)t(k) = \sum_{i=1}^{k} z_i t(i) + (bk - 1)t(k) \\ &\geq T_k - t(k). \end{aligned}$$

This completes the proof of the lemma. □

If $t(i \cdot j) \leq \frac{c}{i} t(j)$, c constant, then the previous lemma yields an approximation algorithm.

Theorem 4. *Let $t(i \cdot j) \leq \frac{c}{i} \cdot t(j)$, c constant. Then the optimal phase-by-phase schedule is optimal up to a factor $1 + \sqrt[6]{\frac{4c^3}{p}}$.*

Proof. Let $\varepsilon > 0$. We will show that the optimal phase-by-phase schedule is optimal up to a factor $1+\varepsilon$ if $p \geq 4 \cdot \frac{c^3}{\varepsilon^6}$ holds.
$n < \varepsilon \cdot p$: We choose the minimal $i \in \mathbb{N}$ with $\frac{p}{i+1} < n \leq \frac{p}{i}$. Hence $t(i)$ is an upper

bound and $t(i+1)$ is a lower bound on the makespan of an optimal schedule. This yields

$$\frac{t(i)}{t(i+1)} \leq \frac{t(i)}{\frac{i}{i+1} \cdot t(i)} = 1 + \frac{1}{i},$$

and we are done.

$n \geq \varepsilon \cdot p$: The coefficients $y_1, \ldots, y_p$ in the definition of T_k define a phase-by-phase schedule. With this schedule we can not handle all jobs. However, due to speed-up property we have at most $i-1$ phases using i processors. So we have at most $\sum_{1 \leq i \leq k}(i-1) = \frac{1}{2}(k-1)k$ phases using more than one processor. If $p \geq \frac{1}{2}k^3$, then the remaining jobs can be executed in exactly one more phase using k processors. For an optimal phase-by-phase schedule with makespan T_{PPS} using at most k processors we have $T_{PPS} \leq T_k + t(k)$. We choose the minimal $k \in \mathbb{N}$ with $k \geq \frac{2c}{\varepsilon^2}$ and get

$$\begin{aligned} T_{PPS} &\leq T_k + t(k) \leq T_p + 2t(k) \leq T_p + \frac{2c}{k} t(1) \\ &\leq T_p + \varepsilon^2 t(1) \leq T_p + \varepsilon \cdot \frac{n}{p} t(1) \leq (1+\varepsilon) \cdot T_p. \end{aligned}$$

This completes the proof. □

Remark 1.

1. Combining Theorems 1, 2 and 4 leads to an ε-approximation algorithm. Use the optimal phase-by-phase algorithm if $p \geq 4 \cdot \frac{c^3}{\varepsilon^6}$ holds and the optimal algorithm from Theorem 1 otherwise.
2. Theorem 4 can be generalized to other time functions fulfilling $t(i) \to 0$ for $i \to \infty$.

Acknowledgment

The authors would like to thank Denis Trystram for Example 1 and Norbert Sensen for part (2) of Theorem 1, and both of them for the fruitful discussions.

References

1. B. Baker, E. Coffman, and R. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.
2. B.S. Baker, D.J. Brown, and H.P. Katseff. A 5/4 algorithm for two-dimensional packing. *Journal of Algorithms*, 2:348–368, 1981.
3. K.P. Belkhale and P. Banerjee. Partitionable independent task scheduling problem. In *Proc. of the 1990 International Conference on Parallel Processing (ICPP'90)*, volume 1, pages 72–75, 1990.
4. J. Blazewicz, M. Machowiak, G. Mounié, and D. Trystram. Approximation algorithms for scheduling independent malleable tasks. In *Proc. of the 7th European Conference on Parallel Computing (Euro-Par'01)*, pages 191–197, 2001.

5. P. Brucker. *Scheduling Algorithms*. Springer Verlag, 1995.
6. W. Fernandez de la Vega and V. Zissimopoulos. An approximation scheme for strip packing of rectangles with bounded dimensions. *Discrete Applied Mathematics*, 82:98–101, 1998.
7. T. Decker. *Ein universelles Lastverteilungssystem und seine Anwendung bei der Isolierung reeller Nullstellen*. Dissertation, 2000.
8. T. Decker and W. Krandick. Parallel real root isolation using the descartes method. In *Proc. of the 6th High Performance Conference (HiPC'99)*, pages 261–268, 1999.
9. T. Decker, T. Lücking, and B. Monien. A 5/4-approximation algorithm for scheduling identical malleable tasks. Technical report, tr-rsfb-02-071, University of Paderborn, 2002.
10. J. Du and Y-T. Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989.
11. H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.
12. M. Garey and R. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975.
13. K. Jansen. Scheduling malleable parallel tasks: An asymptotic fully polynomial-time approximation scheme. In *Proc. of the 10th Annual European Symposium on Algorithms (ESA'02)*, pages 562–573, 2002.
14. K. Jansen and L. Porkolab. Linear time approximation schemes for scheduling malleable parallel tasks. *Algorithmica*, 32(3):507–520, 2002.
15. C. Kenyon and E. Remila. Approximate strip packing. In *Proc. of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 142–154, 1996.
16. S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangular tiling and packing. In *Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'98)*, pages 384–393, 1998.
17. R. Krishnamurti and E. Ma. The processor partitioning problem in special-purpose partitionable systems. In *Proc. of the 1988 International Conference on Parallel Processing (ICPP'88)*, volume 1, pages 434–443, 1988.
18. G. Mounié, C. Rapine, and D. Trystram. Efficient approximation algorithms for scheduling malleable tasks. In *Proc. of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'99)*, pages 23–32, 1999.
19. G. Mounié, C. Rapine, and D. Trystram. A $\frac{3}{2}$-approximation algorithm for independent scheduling malleable tasks. *submitted for publication*, 2001.
20. A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.
21. J. Turek, J.L. Wolf, and P.S. Yu. Approximate algorithms for scheduling parallelizable tasks. In *Proc. of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'92)*, pages 323–332, 1992.
22. B. Veltman, B.J. Lageweg, and J.K. Lenstra. Multiprocessor scheduling with communication delays. *Parallel Computing*, 16:173–182, 1990.

Optimal On-Line Algorithms to Minimize Makespan on Two Machines with Resource Augmentation*

Leah Epstein[1] and Arik Ganot[2]

[1] School of Computer Science, The Interdisciplinary Center, Herzliya, Israel
lea@idc.ac.il
[2] School of Computer Science, Tel-Aviv University, Ramat-Aviv, Tel-Aviv, Israel

Abstract. We study the problem of on-line scheduling on two uniformly related machines where the on-line algorithm has resources different from those of the off-line algorithm. We consider three versions of this problem: preemptive semi-online, non-preemptive on-line and preemptive on-line scheduling. For all these cases we design algorithms with best possible competitive ratios as a function of the machine speeds.

1 Introduction

The problem: We consider scheduling on two uniformly related machines with resource augmentation. Jobs arrive one by one, and each job has to be assigned before the next job arrives.

We consider three different versions: preemptive semi-online scheduling, non-preemptive online scheduling and preemptive online scheduling, where *semi-online* means that jobs must arrive in a decreasing order of size, and preemptive means that a job may be cut into a few pieces and scheduled on both machines, but two parts of the same job may not be scheduled concurrently on two different machines. This is allowed to both the on-line algorithm and the off-line algorithm.

The measure: A schedule is measured by *makespan*, i.e. the maximum completion time of any machine. An on-line algorithm is measured by its *competitive ratio*, which is defined $C = \sup_{\sigma} \left\{ \frac{ALG(\sigma)}{OPT(\sigma)} \right\}$, where σ is a list of jobs and $ALG(\sigma)$ and $OPT(\sigma)$ are the makespans of the schedules created by the on-line algorithm and the optimal off-line algorithm for that list of jobs.

Preliminaries: We consider cases where the resources may be *augmented*, i.e., the optimal off-line algorithm has two uniformly related machines with possibly different speeds, which may be faster or slower than the speeds of the machines of the on-line algorithm. In the general case, this means that the on-line algorithm has a slow machine with speed a and a fast machine with speed b, and the off-line algorithm has a slow machine with speed A and a fast machine with speed B.

* Research supported in part by the Israel Science Foundation (grant no. 250/01).

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 109–122, 2004.
© Springer-Verlag Berlin Heidelberg 2004

However, with no loss of generality we may assume the on-line algorithm has a slow machine with speed $1/s$ ($s \geq 1$) and a fast machine with speed 1, and the off-line algorithm has a slow machine with speed $1/q$ ($q \geq 1$) and a fast machine with speed 1. Note that a machine of speed $1/z$ is faster for smaller values of z.

This assumption can be made since a schedule with makespan x on machines with speeds a, b will have a makespan x/a on machines with speeds 1, b/a. In particular this is true for the optimal on-line schedule, thus it is possible to assume, without loss of generality, that the on-line algorithm has speeds $1, 1/s = a/b$. Similarly, it is possible to assume, without loss of generality, that the off-line algorithm has speeds $1, q = A/B$. We use the parameters q and s throughout the paper.

The load of a job of processing time p on a machine of speed $1/z$ is pz. We define *load* of a machine as the total load of the jobs (or parts of jobs) that are scheduled on that machine plus the sum of times when the machine is idle. The *weight* of a machine is the sum of the processing times of jobs (or part of jobs) that are scheduled on that machine.

We slightly abuse the notation by identifying jobs with their sizes (processing times)

Our results: In this paper we find the exact competitive ratio functions for all three versions we consider, those are functions of s and q. For the preemptive cases, we show they are valid for both a deterministic version and a randomized version. We give deterministic algorithms which achieve these competitive ratios.

We show that the competitive ratio of preemptive semi-online scheduling is 1 for $\{q \leq 3, s \leq \frac{2q}{3}\}$ or for $\{q \geq 3, s \leq \frac{q+1}{2}\}$, $\frac{3s(q+1)}{2q+3sq}$ for $\{q \leq 3, \frac{2q}{3} \leq s \leq 2\}$ and $\{q < 3, s > 2\}$, and $\frac{2s(q+1)}{q+1+2sq}$ for $\{q \geq 3, s > \frac{q+1}{2}\}$. For the last two cases we use idle time and prove that it is necessary.

Moreover, we show that the competitive ratio of non-preemptive online scheduling is $\min\left\{\frac{(1+2q)s}{(s+1)q}, \frac{q+1}{q}\right\}$ and the competitive ratio of preemptive on-line scheduling is $\frac{(q+1)^2 s}{q(1+s+sq)}$.

Previous work

On-line scheduling: Scheduling on identical and uniformly related was studied since 1966 when Graham introduced his greedy algorithm "List Scheduling" for identical machines [20]. The competitive ratio of that algorithm is $2-1/m$. Since then the was a sequence of improvements on the performance of algorithms [4, 25, 1] and a sequence of lower bounds [5, 1, 19], and the best current results are an algorithm of competitive ratio 1.9201, designed by Fleischer and Wahl [16] and a lower bound of 1.88 given by Rudin [28]. For uniformly related machines there exist constant competitive algorithms, see [2, 6]. The lowest upper bound known for the competitive ratio is 5.828 whereas the lower bound is 2.438. For two and three identical machines, [15] showed that the greedy algorithm has the best possible competitive ratio. The tight result for two uniformly related machines (without resource augmentation) is folklore and is given in [13]. The competitive ratio is $\min\{\frac{2q+1}{q+1}, 1 + \frac{1}{q}\}$ for this problem, achieved by a greedy algorithm which assigns a job to the machine which would finish it first.

Semi-online scheduling: Graham [21] also analyzed the greedy algorithm on identical machines for the case where sizes (processing times) of jobs are non-increasing getting the result $4/3-1/(3m)$. The results for two and three machines are 7/6 and 11/9. Seiden, Sgall and Woeginger [30] showed lower bounds of 7/6 and 1.18046 for two and three machines respectively. Note that the bound for two machines is tight, which means that the greedy algorithm has the best possible competitive ratio for the problem.

Most of the study for semi-online scheduling on non-identical machines involves a study of the greedy algorithm. For two machines, Mireault, Orlin and Vohra [27] gave a complete analysis of the greedy algorithm as a function of the speed ratio. They show that the interval $q \geq 1$ is partitioned into nine intervals, and introduced a function which gives the competitive ratio in each interval. The reference [18] shows that for any speed ratio, the performance ratio of the greedy algorithm is at most $\frac{1}{4}(1+\sqrt{17}) \approx 1.28$. This was generalized in [11]. In that paper, a complete analysis of the best competitive ratio as a function of the speed ratio was given. Here there are already fifteen intervals. In most intervals the greedy algorithm turns out to be the best algorithm, but there are several intervals where other algorithms are designed, and were shown to perform better and to have the best possible competitive ratios. For a general setting of m uniformly related machines, Friesen [17] showed that the overall approximation ratio of the greedy algorithm is between 1.52 and $\frac{5}{3}$. Dobson [9] claimed to improve the upper bound to $\frac{19}{12} \approx 1.58$. Unfortunately, his proof does not seem to be complete.

On-line preemptive scheduling: There are several tight results for preemptive *on-line* scheduling, in all cases the lower bounds hold also for randomized algorithms. Chen, Van Vliet and Woeginger gave a preemptive optimal algorithm and a matching lower bound for *identical machines* [8]. The competitive ratio of the algorithm is $m^m/(m^m-(m-1)^m)$. Seiden [29] also gave such an optimal algorithm. His algorithm, has a lower makespan than the one in [8] in many cases, however (naturally) it has the same competitive ratio. A lower bound of the same value was given independently by Sgall [31]. For *two uniformly related machines*, the tight competitive ratio is known to be $(q+1)^2/(q^2+q+1)$, given in [13] and independently in [33]. Those results are extended for a class of m uniformly related machines with *non-decreasing speed ratios* in [10]. The tight bound in that case is a function of all the speeds. A lower bound of 2 on the competitive ratio was given already in [14].

Semi-online preemptive scheduling: Seiden, Sgall and Woeginger [30] studied semi-online preemptive scheduling on *identical machines*. They showed that the most difficult case among non-increasing sequences is a sequence of unit size jobs. They gave a complete solution, where the competitive ratio is a function of the number m of machines.

The problem of preemptive semi-online scheduling on two uniformly related machines without resource augmentation was studied by Epstein and Favrholdt [12], who proved a competitive ratio of $\max\{\frac{(3(q+1)}{3q+2}, \frac{2q(q+1)}{2q^2+q+1}\}$ where $1/q$ is the speed of the slow machine. They also proved that idle time is necessary for $q > 2$.

Resource augmentation: This is a generalization of competitive analysis. Resource augmentation was introduced by Kalyanasundaram and Pruhs [24]. They studied certain scheduling problems which are known to have unbounded competitive ratio. They showed that it is possible to attain a good competitive ratio if the machines of the on-line algorithm are slightly faster than the machines of the off-line algorithm.

Resource augmentation has been applied to a number of problems. It was already used in the paper where the competitive ratio was introduced [32]. In that paper, the performance of paging algorithms was studied, where they have a larger cache memory than that of the optimal off-line algorithm. In several machine scheduling and load balancing problems [7, 23, 26, 3], the effect of adding more or faster machines has been studied.

2 Preemptive Semi-online Scheduling

In this section we consider semi-online preemptive scheduling on two uniformly related machines. Recall that jobs arrive sorted in non-increasing order of processing times. In the current section $c(q, s)$ denotes the optimal competitive ratio for this problem. We prove the following theorem.

Theorem 1. *The optimal competitive ratio of deterministic or randomized semi-online preemptive algorithm is:*

$$c(q,s) = \begin{cases} 1 & \text{for } q \leq 3 \text{ and } s \leq \frac{2q}{3} \\ 1 & \text{for } q \geq 3 \text{ and } s \leq \frac{q+1}{2} \\ \frac{3s(q+1)}{2q+3sq} & \text{for } q \leq 3 \text{ and } \frac{2q}{3} \leq s \leq 2 \\ \frac{3s(q+1)}{2q+3sq} & \text{for } q < 3 \text{ and } s > 2 \\ \frac{2s(q+1)}{q+1+2sq} & \text{for } q \geq 3 \text{ and } s > \frac{q+1}{2} \end{cases}$$

In the last two cases, only an algorithm which uses idle time can achieve this competitive ratio.

See Figure 1 for the five cases.

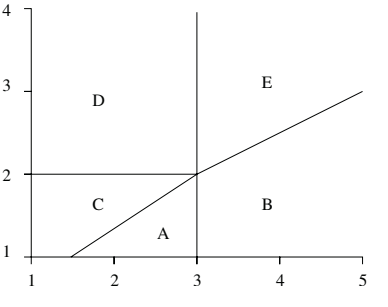


Fig. 1. The partition to areas according to the five cases

We first prove the lower bound and then design the algorithms. The only algorithms which use idle time are for the cases where it is necessary. Note that due to the resource augmentation, there are two areas where the slow machine of the on-line algorithm is relatively fast and therefore an "optimal" algorithm can be achieved. In those cases, adding speed to the on-line algorithm allows us to overcome the on-line restriction.

2.1 Lower Bound

We start with two theorems that appeared in the literature and are useful for proving our results.

The first theorem deals with the optimal preemptive schedule. Unlike the non-preemptive case there is a simple formula for calculating it. The theorem is a special case of a theorem given in [22].

Theorem 2. *Given a set of jobs of total weight W, the largest of which has size ℓ, the makespan of the optimal offline algorithm (whose machine speeds are* 1 *and* $1/q$*) is* $\max\{\ell, \frac{Wq}{q+1}\}$.

The second theorem gives a "recipe" of computing lower bounds for randomized on-line and semi-online algorithms. Similar theorems were given in [14, 10, 12] and are adaptations of a theorem from [31].

Theorem 3. *Consider a sequence of at least two jobs, where J_{n-1}, J_n are the last two jobs. Let $OPT(J_i)$ be the preemptive optimal off-line cost after the arrival of J_i. The competitive ratio of any preemptive randomized on-line algorithm is at least* $\frac{Ps}{OPT(J_{n-1})+sOPT(J_n)}$.

Proof. Fix a sequence of random bits used by the algorithm. Let $ONL(J_i)$ be the makespan of the preemptive on-line algorithm after scheduling J_i. Let $T_1 = ONL(J_n)$, and let T_2 be the last time which the on-line algorithm has a job scheduled on both machines. Since a job cannot be processed concurrently on both machines, we get $ONL(J_{n-1}) \geq T_2$.

Suppose c is the competitive ratio of the on-line algorithm. Let P be the total size of all the jobs. Since only one machine can be non-idle during the time period between T_2 and T_1, we get $P \leq (1 + \frac{1}{s})T_2 + (T_1 - T_2) = T_1 + \frac{T_2}{s}$. Taking the expectation we get
$P \leq E[T_1] + \frac{E[T_2]}{s} \leq E[ONL(J_n)] + \frac{E[ONL(J_{n-1})]}{s} \leq cOPT(J_n) + \frac{cOPT(J_{n-1})}{s}$.
Hence, $c \geq \frac{Ps}{OPT(J_{n-1})+sOPT(J_n)}$.

Note that we did not use the assumption that the jobs arrive at a decreasing order of size, thus this theorem applies for the on-line case as well.

Lemma 1. *The competitive ratio of any on-line preemptive randomized algorithm is at least:*
$\max\left\{r_1 = \frac{2s(q+1)}{q+1+2sq}, r_2 = \frac{3s(q+1)}{2q+3sq}, 1\right\}$.

Proof. Consider three sequences consisting of 1, 2 and 3 unit size jobs. For one job, no algorithm can do better than scheduling all of it on the fast machine at time 0, achieving a competitive ratio of 1. For sequences of 2 and 3 jobs, we get the lower bound by applying the above theorem.

Lemma 2. *For the cases* $\{q < 3\ ,\ s > 2\}$ *and* $\{q \geq 3, s > \frac{q+1}{2}\}$ *an on-line algorithm which does not use idle time cannot achieve a competitive ratio of* r_2 *and* r_1*, respectively.*

Proof. Consider a sequence of two unit jobs. Set $i = 1$ for $q \geq 3$, $i = 2$ for $q < 3$. Under our assumptions, the first job cannot be assigned to the slow machine, since in this case $OPT = 1$. This would cause the competitive ratio to become s, which breaches our competitive ratio, since
$r_1 - s = \frac{2s(q+1)}{q+1+2sq} - s = s\frac{q+1-2sq}{q+1+2sq} < 0$ since $s > \frac{q+1}{2}$ in this case, and
$r_2 - s = \frac{3s(q+1)}{2q+3sq} - s = s\frac{q+3-3sq}{q(2+3s)} < 0$ since $3sq > 6$ and $q+3 \leq 6$. Thus the first job must be scheduled on the fast machine at time 0. For two jobs, $OPT = \frac{2q}{q+1}$, thus the second job must be scheduled after time 1 and no later than time $\frac{2q}{q+1}r_i$ on the fast machine, which means at least $1 - (\frac{2q}{q+1}r_i - 1)$ must be scheduled on the slow machine. For *both* these cases, this schedule is illegal since $s\left(2 - \frac{2q}{q+1}r_i\right) > 1$ due to the following.

For r_1, $2s - 1 - s\frac{2q}{q+1}r_1 = \frac{2s-q-1}{q+1+2sq} > 0$ for $s > \frac{q+1}{2}$, and for r_2, $2s - 1 - s\frac{2q}{q+1}r_2 = \frac{s-2}{2+3s} > 0$ for $s > 2$. This implies that the second job is scheduled concurrently on both machines.

2.2 Algorithms without Idle Time

In this section we present algorithms for the three cases in which the creation of idle time in the schedules can be avoided. By the previous section, only three such cases may exist: $\{s \leq \frac{2q}{3}, q \leq 3\}$, $\{\frac{2q}{3} < s < 2, q \leq 3\}$ and $\{s \leq \frac{q+1}{2}, q > 3\}$.

Lemma 3. *For* $\max\{1, \frac{2q}{3}\} \leq s \leq 2, q \leq 3$*, there exists an on-line algorithm of competitive ratio* $r_2 = \frac{3s(q+1)}{2q+3sq}$ *which does not use idle time.*

Proof. The first job J_1 is scheduled on the fast machine. without loss of generality, we may assume it has size 1.

As long as the total size of jobs does not exceed $1 + \frac{1}{q}$, $OPT = 1$. Thus, we can use the interval 1 to r_2 on the fast machine without violating the competitive ratio. For the next jobs we use that interval first, and when it is full we go on to the time interval $[0, s\left(1 + \frac{1}{q} - r_2\right)]$ on the slow machine. When these two intervals are filled, the total size of the jobs scheduled is $1 + \frac{1}{q}$. At this point, some job may be only partially assigned. Denote this job J_p. Since $(1 + \frac{1}{q} - r_2)s = \left(1 + \frac{1}{q} - 3s\frac{q+1}{2q+3sq}\right)s = 2s\frac{q+1}{q(2+3s)} = \frac{2s}{2+3s}\frac{q+1}{q} \leq \frac{4}{8} \cdot \frac{2}{1} \leq 1$, the load on the slow machine does not exceed 1, hence even if until this point some other job was split between the two machines, its two parts do not overlap in time.

In this situation, the ratio of the loads is 3:2. From now on, we keep this ratio, so that the fast machine is always more loaded. Any arriving job of size x is split into two pieces of size $\frac{3sx}{3s+2}$ (fast machine) and $\frac{2x}{3s+2}$ (slow machine). For the case that the total sum of sizes P is large enough ($P \geq 1+\frac{1}{q}$), $OPT = \frac{qP}{q+1}$. The on-line load on the fast machine (which is larger) is $\frac{3sP}{3s+2}$, and the competitive ratio is r_2. This completes the description of the algorithm. To complete the proof we need to show the following:

a. The second part of J_p is scheduled properly.
b. Any future job J is scheduled properly.

Proof of a – case 1: J_p was the second job to arrive: Since it is scheduled on the fast machine only after time 1, we need only to show that on the slow machine, it will be completed no later than time 1. Let x be the size of the second part of J_p. We need to schedule $\frac{2x}{3s+2}$ of the job (which consumes $\frac{2xs}{3s+2}$ units of time on the slow machine). The size of the second job is at most 1, and $\frac{1}{q}$ of it was scheduled, so $x \leq 1-\frac{1}{q}$. Thus, after scheduling J_p, we get that the load on the slow machine is $\frac{2s}{3s+2} \cdot \frac{q+1}{q} + \frac{2xs}{3s+2} \leq \frac{2s}{3s+2}\left(1+\frac{1}{q}+x\right) \leq \frac{4s}{3s+2} \leq 1$.

Proof of a – case 2: J_p was the third job to arrive or later: If J_p does not have its first part scheduled on the fast machine, then this case is similar to a case here the first part and the second part are two different jobs, whose correctness is proven in the next section. Otherwise, $J_p < r_2 - 1$, since it must be smaller than the second job, and the second job must be scheduled entirely on the first machine by the definition of the algorithm. In this case, scheduling it on the slow machine will cause a load of no more than $(r_2-1)s = \frac{3s^2-2sq}{2q+3sq} \leq \frac{3s^2-2s}{2+3s} \leq 1$ (for $s \leq 2$). Thus, the part scheduled on the slow machine does not intersect J_2, and thus it does not intersect J_3.

Proof of b: We need to show that the interval on the slow machine is enough to schedule $\frac{2x}{3s+2}$. Let P be the total size of previous jobs. Then $\frac{3s}{3s+2}P$ is the load of the fast machine and $\frac{2s}{3s+2}P$ is the load of the slow machine just before scheduling J. The extra load that J causes is $\frac{2xs}{3s+2}$. Hence we need $\frac{Ps}{3s+2} \geq \frac{2sx}{3s+2}$. This clearly holds since $x \leq \frac{P}{2}$ because J is at least the third job.

The proofs on the next two lemmas are omitted.

Lemma 4. *For $s \leq \frac{q+1}{2}, q \geq 3$, there exists an on-line algorithm of competitive ratio 1 which does not use idle time.*

Lemma 5. *For $s \leq \frac{2q}{3}, q \leq 3$, there exists an on-line algorithm of competitive ratio 1 which does not use idle time.*

2.3 Algorithms with Idle Time

In this section we present algorithms for cases which were previously proven to require idle time to reach our desired competitive ratio. Two such cases exist: $\{q < 3, s > 2\}$, for which we will prove a competitive ratio of $r_2 = \frac{3s(q+1)}{2q+3sq}$, and $\{q \geq 3, s > \frac{q+1}{2}\}$ for which we will prove a competitive ratio of $r_1 = \frac{2s(q+1)}{q+1+2sq}$.

Lemma 6. *For every q there exists an on-line preemptive algorithm with competitive ratio of* $\max\left\{\frac{2s(q+1)}{q+1+2sq}, \frac{3s(q+1)}{2q+3sq}\right\}$.

Proof. We define the algorithm. Without loss of generality, we assume the first job has size 1. Let $r = \frac{3s(q+1)}{2q+3sq}$ if $q < 3$, and $\frac{2s(q+1)}{q+1+2sq}$ otherwise. The first job is cut into two pieces: one of size $\frac{r-1}{s-1}$ which is scheduled on the slow machine until time r, and the other of size $\frac{s-r}{s-1}$ which is scheduled on the fast machine starting from time 0. From now on, future jobs are scheduled on the first (possibly idle) time on the machine they are scheduled on. If a job is scheduled on the slow machine and there is no idle time left, it will be scheduled after time r.

As in previous algorithms, as long as the total size of the jobs does not exceed $1 + \frac{1}{q}$, these jobs are scheduled on the fast machine until time r, and then on the slow machine. When the total size of the jobs is $1 + \frac{1}{q}$, there is still idle time on the slow machine - by the definition of the algorithm, the idle time ends when the total size of the jobs is $r(1 + \frac{1}{s})$, which is greater than $1 + \frac{1}{q}$, Since $r(1 + \frac{1}{s}) - 1 - \frac{1}{q} \geq \min\{\frac{q^2-1}{(q+1+2sq)q}, \frac{q+1}{q(2+3s)}\} \geq 0$ for $q \geq 1$. The loads on the machines, not including idle time, is r on the fast machine and $s\left(1 + \frac{1}{q} - r\right)$ on the slow machine. The ratio of the loads is $\frac{3}{2}$ if $q \leq 3$, and $\frac{2}{1+\frac{1}{q}}$ if $q > 3$. Since even after the first job $OPT \geq 1$, the competitive ratio is maintained up to this point.

We continue by keeping the ratio between the loads. For $q \leq 3$ this means cutting any job of size x into two pieces, one of size $\frac{3sx}{3s+2}$, which is assigned to the first violating time on the fast machine, and the other of size $\frac{2x}{3s+2}$, which is assigned to the first available time on the slow machine. For $q > 3$ this means cutting any job of size x into two pieces, one of size $\frac{2xsq}{q+1+2sq}$ which is assigned to the first available time on the fast machine, and the other of size $\frac{x(q+1)}{q+1+2sq}$ which is assigned to the first available time on the slow machine. Note that in both cases it is possible for the assignment of some job to take up the idle time on the slow machine, in which case the piece assigned to the slow machine is cut into two pieces, and is continued after time r.

Again we need to prove the leftover on the job for which the total size reached $1+\frac{1}{q}$ is scheduled properly, and that any future job J are assigned properly. Since the free time before time r on both machines is disjoint after scheduling the first job, there is no difference between a job and the leftover of a job.

Case A: $q \leq 3$:
Let P be the total size of previous jobs (such that $P \geq 1 + \frac{1}{q}$). Let x be the size of job J. Then $\frac{3s}{3s+2}P$ is the load of the fast machine and $\frac{2s}{3s+2}P$ is the load of the slow machine before scheduling J. Scheduling J will cause an extra load of $\frac{2xs}{3s+2}$ on the slow machine. Hence, we need to show that $\frac{Ps}{3s+2} \geq \frac{2xs}{3s+2}$. This clearly holds if is at least the third job, which implies $x \leq \frac{P}{2}$. Otherwise, J is a leftover and then $x \leq 1 - \frac{1}{q}$. For $q \leq 3$, this implies that $x \leq 1 - \frac{1}{q} \leq \frac{1}{2}(1 + \frac{1}{q})$ $\leq \frac{P}{2}$.

Case B: $q > 3$:
Let P be the total size of previous jobs (such that $P \geq 1+\frac{1}{q}$). Let x be the size of job J. Then $\frac{2qs}{2qs+q+1}P$ is the load of the fast machine and $\frac{(q+1)s}{2qs+q+1}P$ is the load of the slow machine before scheduling J. Scheduling J will cause an extra load of $\frac{s(q+1)x}{q+1+2qs}$ on the slow machine. Hence, we need to show that $\frac{(q+1)xs}{q+1+2qs} \leq \frac{(q-1)sP}{q+1+2qs}$, which is implied by $x \leq \frac{q-1}{q+1}P$. If J is the third job or later then $x \leq \frac{P}{2} \leq \frac{q-1}{q+1}P$. Otherwise, J is a leftover and $x \leq 1-\frac{1}{q}$. For $q > 3$, this implies
$x \leq 1-\frac{1}{q} = \frac{(q-1)(q+1)}{q^2+q} \leq \frac{q-1}{q+1} \cdot \frac{q+1}{q} \leq \frac{q-1}{q+1}P$.

3 Non-preemptive On-Line Scheduling

In this section we study the problem of deterministic non-preemptive scheduling with respect to makespan on two machines. There is no restriction on the arriving jobs. Since preemption is not allowed, we can assume that the algorithms cannot use idle time, as idle time does not improve the situation. In the current section $c(q,s)$ denotes the optimal competitive ratio for this problem.

We prove the following theorem.

Theorem 4. *The optimal competitive ratio of deterministic on-line non-preemptive algorithm is* $c(q,s) = \frac{(1+2q)s}{(s+1)q}$ *for* $s < \frac{q+1}{q}$ *and* $c(q,s) = \frac{q+1}{q}$ *for* $s \geq \frac{q+1}{q}$.

Note that $1 \leq c(q,s) \leq 2$.

3.1 Upper Bound

We define the algorithm LIST. Schedule each job on the machine where it will be completed first.

Lemma 7. *LIST has a competitive ratio of* $\min\left\{\frac{(1+2q)s}{(s+1)q}, \frac{q+1}{q}\right\}$.

Proof. Consider the time just before the arrival of the job which determines the makespan. Let x be the weight of the fast machine of the on-line algorithm, and let y be the weight on the slow machine of the on-line algorithm at this point. Let ℓ be the size of the next job. Denote OPT the makespan of the schedule created by the optimal off-line algorithm, ALG the makespan of the schedule created by LIST, and $c = \frac{ALG}{OPT}$. We get $OPT \geq \frac{x+y+l}{1+\frac{1}{q}} = \frac{q(x+y+l)}{1+q}$ and $OPT \geq l$. The on-line algorithm will assign the next job to the machine where it will finish first. Thus $c(ONL) \leq x+l$ and $c(ONL) \leq (y+l)s$. We have $(s+1)c(ONL) \leq s(x+y+2l) \leq s(x+y+l)+sl \leq \frac{s(1+q)}{q}OPT + sOPT$. Thus $c(ONL) \leq OPT \cdot \frac{s(1+q)+qs}{(s+1)q} \rightarrow c \leq \frac{s(1+2q)}{q(s+1)}$. Moreover, LIST is definitely no worse than scheduling every job on the fast machine - hence, $c \leq \frac{q+1}{q}$. Note that $c = \frac{q+1}{q}$ for every $s \geq \frac{q+1}{q}$. This value varies between 1 and 2, and is equal to $\phi \approx 1.618$ for s=q.

3.2 Lower Bound

We prove that no deterministic on-line algorithm can perform better than LIST.

Lemma 8. *No deterministic on-line algorithm can have a better competitive ratio than* $c = \min\left\{\frac{(1+2q)s}{(s+1)q}, \frac{q+1}{q}\right\}$.

Proof. First, we prove the lemma for $s \geq \frac{q+1}{q}$. In this case, $c = \frac{q+1}{q}$. We use a sequence of two jobs: 1, q. Denote the on-line algorithm ALG, and an optimal off-line algorithm OPT. If ALG schedules the first job on the slow machine, it gets a competitive ratio of $s \geq \frac{q+1}{q}$. Thus in order to maintain a competitive ratio of less than $\frac{q+1}{q}$ it must schedule the first job on the fast machine. In this case, it performs no better then $q+1$ on the second job since scheduling it on the fast machine gives a makespan of $q+1$, and scheduling it on the slow machine gives a makespan of $qs \geq q+1$, while OPT easily achieves a makespan of q.

For $s < \frac{q+1}{q}$, Set $\alpha = \frac{q+1-qs}{s-q+qs}$. Note that since $s < \frac{q+1}{q}$, $\alpha > 0$. We start with presenting the algorithm a job of size 1. We have the following cases:

Case 1: The first job is assigned to the slow machine.
Set $T_i = (1+\frac{1}{\alpha})^i$. Let j be an integer number such that $T_j \leq 1+q, T_{j+1} > 1+q$. Such a number j must exist since $T_0 = 1$ and $\lim_{n\to\infty} T_n = \infty$. In this case, we proceed with the following sequence: $b_0 = 1$ (The job which was presented at first), $b_1 = \frac{1}{\alpha}, b_i = \frac{1}{\alpha}(1+\frac{1}{\alpha})^{i-1}$ for all $i \geq 2$. Note that $T_i = \sum_{n=0}^{i} b_n$, for all $i \geq 0$. We give jobs from this sequence until some job b_t is scheduled on the fast machine, or until job $b_t = b_{j+1}$ is scheduled (on one of the machines), i.e. in total $j+2$ jobs are scheduled. The following cases may occur:

Case 1a: $t < j+1$ and there is at least one job on each machine**:**
After the first time the algorithm scheduled a job on the fast machine, we present a job of size qT_t. The off-line algorithm achieves a makespan of qT_t (scheduling all but the last job on the slow machine, and the last job on the fast machine). The on-line algorithm can either schedule that job on the slow machine or to the fast machine. If it is scheduled on the slow machine, we get a competitive ratio of $\frac{sT_{-1}+sqT}{qT} = \frac{s(1+\frac{1}{})^{-1}+sq(1+\frac{1}{})}{q(1+\frac{1}{})} = \frac{s}{(1+\frac{1}{})q} + s = \frac{s(q+1-qs+qs+q)}{q(s+1)} = \frac{(1+2q)s}{(s+1)q}$
Otherwise, it is scheduled on the fast machine, and the competitive ratio is $\frac{b+qT}{qT} = \frac{\frac{1}{}(1+\frac{1}{})^{-1}+q(1+\frac{1}{})}{q(1+\frac{1}{})} = 1 + \frac{1}{q(1+\alpha)} = 1 + \frac{1}{q(1+\frac{1+-}{+-})} = \frac{qs+s-q}{q(s+1)} + 1 = \frac{2qs+s}{q(s+1)} = \frac{s(1+2q)}{(s+1)q}$.

Case 1b: $t = j+1$ and all jobs are on one machine**:**
We will show that either after $j+1$ jobs or after $j+2$ jobs, the competitive ratio is at least c. We denote by OPT_k, ALG_k the makespans after scheduling job b_k of the optimal algorithm and the on-line algorithm, respectively. In these two cases ($k = j, j+1$), OPT can put the first job on the slow machine and all the next jobs on the fast machine, achieving a makespan of $OPT_j \leq q$ $OPT_{j+1} \leq T_j - 1$. The on-line algorithm places all jobs on the slow machine, achieving a makespan $ALG_j = sT_j, ALG_{j+1} = sT_{j+1}$. If $\frac{sT}{q} \geq \frac{s(1+2q)}{q(s+1)}$ the proof is complete

since $\frac{ALG}{OPT} \geq c$. Otherwise, $T_j < \frac{1+2q}{s+1}$ and $T_{j+1} = T_j(1+\frac{1}{\alpha}) < \frac{1+2q}{s+1}(1+\frac{1}{\alpha}) = \frac{1+2q}{q+1-qs}$. Thus, the competitive ratio is at least $\frac{sT_{+1}}{T_{+1}-1} = s\left(1+\frac{1}{T_{+1}-1}\right) \geq s\left(1+\frac{1}{\frac{1+2}{+1-}-1}\right) = s\frac{2q+1}{(s+1)q}$.

Case 2: The first job is scheduled on the fast machine.
Set $S_i = (1+\alpha)^i$. Let j be an integer number such that $S_j \leq 1+q, S_{j+1} > 1+q$. Such a value j must exist since $S_0 = 1$ and $\lim_{n\to\infty} S_n = \infty$. In this case, we proceed with the following sequence: $a_0 = 1$ (The job which was presented at first), $a_1 = \alpha, a_i = \alpha(1+\alpha)^{i-1}$ for all $i \geq 2$. Note that $S_i = \sum_{n=0}^{i} a_n$. We give jobs from this sequence until some job is scheduled on the slow machine, or until $j+2$ jobs are scheduled. Let k be the index of the last job. The following cases may occur:

Case 2a: $k < j+1$ and there is at least one job on each machine
After the first time the algorithm scheduled a job on the fast machine, we give the on-line algorithm a job of size qS_k. The off-line algorithm achieves a makespan of qS_k (scheduling all but the last job on the slow machine, and the last job to the fast machine). An on-line algorithm can either schedule the last job on the fast machine or on the slow machine.

If it is scheduled on the fast machine, the competitive ratio is at least $\frac{S_{-1}+qS}{qS} = \frac{(1+\alpha)^{-1}+q(1+\alpha)}{q(1+\alpha)} = 1+\frac{1}{q(1+\alpha)} = 1+\frac{1}{q(1+\frac{1+-}{+-})} = \frac{qs+s-q}{q(s+1)}+1 = \frac{2qs+s}{q(s+1)} = \frac{s(1+2q)}{(s+1)q}$
If J is scheduled on the slow machine, the competitive ratio is at least $\frac{sa_{+}+sqS}{qS} = \frac{s\alpha(1+\alpha)^{-1}+sq(1+\alpha)}{q(1+\alpha)} = \frac{s\alpha}{(1+\alpha)q}+s = \frac{s+qs-qs^2}{(s+1)q}+s = s\frac{2q+1}{(s+1)q}$

Case 2b: $k = j+1$ and all jobs are on one machine
We show that after either $j+1$ jobs or $j+2$ jobs, the competitive ratio is at least c. We denote by OPT_t, ALG_t the makespans after scheduling job a_t $(t = k, k+1)$ of the optimal algorithm and the on-line algorithm, respectively. In this case, OPT can put the first job on its slow machine and all the next jobs on its fast machine, achieving a makespan of $OPT_j \leq q$, and $OPT_{j+1} \leq S_{j+1} - 1$. The on-line algorithm has scheduled the first $j+2$ jobs on the fast machine, achieving a makespan $ALG_j = S_j$, $ALG_{j+1} = S_{j+1}$. If $\frac{Sj}{q} \geq \frac{s(1+2q)}{q(s+1)}$ the proof is complete since $\frac{ALG}{OPT} \geq c$. Otherwise, $S_j < \frac{s(1+2q)}{s+1}$ and $S_{j+1} = S_j(1+\alpha) < \frac{s(1+2q)}{s+1}(1+\alpha) = (1+2q)\frac{s}{s-q+qs}$. Thus, the competitive ratio is at least $\frac{S_{+1}}{S_{+1}-1} = 1+\frac{1}{S_{+1}-1} \geq 1+\frac{1}{\frac{(1+2)}{-+}-1} = s\frac{2q+1}{(s+1)q}$

4 Preemptive On-Line Scheduling

We study the problem of deterministic and randomized scheduling with respect to makespan on two machines. Idle time is allowed but our algorithms do not use it.

In this section we let $r(s,q)$ denote the competitive ratio function. We prove the following theorem:

Theorem 5. *The optimal competitive ratio of any deterministic or randomized on-line preemptive algorithm is* $r(s,q) = \frac{(1+q)^2 s}{(1+s+sq)q}$.

Note that for all s and q, $1 \leq r \leq 2$ holds.

4.1 Lower Bound

Lemma 9. *No deterministic or randomized on-line algorithm can achieve a competitive ratio better than* $r = \frac{(1+q)^2 s}{(1+s+sq)q}$.

Proof. We get this from theorem 1 stated in section 2, by using a sequence of sand with volume 1, followed by a job with size q. The optimal algorithm achieves a makespan of $\frac{1}{1+\frac{1}{s}}$ on the sand. This is done by assigning an amount of $\frac{1}{1+\frac{1}{s}}$ of the sand to the fast machine and the rest which is an amount of $\frac{1}{q\left(1+\frac{1}{s}\right)}$ of it to the slow machine. A makespan of q on the entire sequence is achieved by assigning all the sand to the slow machine and the other job to the fast machine. We get $c \geq \frac{Ps}{OPT(J_{-1})+sOPT(J)} = \frac{(q+1)s}{\frac{1}{1+\frac{1}{s}}+qs} = \frac{(q+1)^2 s}{q+sq(q+1)} = \frac{(q+1)^2 s}{q(1+s+qs)}$

4.2 Upper Bound

We use the following notation. A new (arriving) job has size ℓ. The makespan of the optimal off-line algorithm before scheduling ℓ is OPT. The load of the fast machine of the on-line algorithm before scheduling ℓ is L_1, and the load of the slow machine of the on-line algorithm before scheduling ℓ is L_2. The total weight of all the jobs before ℓ is $w = L_1 + \frac{L_2}{s}$. Analogously, L_1', L_2', w' and OPT' are the load on the fast machine of the on-line algorithm, the load on the slow machine of the on-line algorithm and the total weight of the presented jobs and the makespan of the optimal off-line algorithm after scheduling ℓ.

Lemma 10. *For a given sequence of job* $a_1, a_2, \ldots, a_n$, *Let* $w = \sum_{i=1}^{n} a_n$. *If there exists some i for which* $a_i > q(w - a_i)$, *then it is unique and the competitive ratio of the optimal off-line algorithm is* a_i. *Otherwise, the makespan of the optimal off-line algorithm is* $w/(1 + 1/q)$.

Proof. If a_i exists then it must be unique since $a_i > qw/(q+1) \geq w/2$. The bound on the makespan of the optimal off-line algorithm follows from Theorem 2.

We define algorithm F: Whenever a new job ℓ arrives, schedule a part as large as possible of it within the time interval $[L_1, rOPT']$ on the fast machine starting from L_1, and process the rest (if necessary) on the slow machine starting from L_2.

The proofs on the next three lemmas are omitted.

Lemma 11. *At any state of algorithm F,* $L_2 \leq \frac{L_1}{q+1}$.

Lemma 12. *For a given* w, ℓ *the smallest available space for* ℓ *is achieved when* $L_2 = \frac{L_1}{q+1}$.

Lemma 13. *At every state of algorithm F, the assignment is legal.*

5 Conclusion

We gave tight bounds for three models. We did not study the on-line non-preemptive model. In [11] where this model is studied, the interval $[1, \infty)$ of the speed ratio between the two machines is split into 15 intervals, each of which stands for a totally different case. The proofs are very technical and therefore it seems that an extension to the resource augmented model would be too complicated. There are certainly many other two machine models which are interesting to extend.

References

1. S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459-473, 1999.
2. J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *Journal of the ACM*, 44:486–504, 1997.
3. N. Bansal, K. Dhamdhere, J. Könemann, and A. Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, pages 260–270, 2003.
4. Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51(3):359–366, 1995.
5. Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Information Processing Letters*, 50:113–116, 1994.
6. P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35, 2000.
7. P. Berman and C. Coulston. Speed is more powerful than clairvoyance. *Nordic Journal of Computing*, 6(2):181–193, 1999.
8. B. Chen, A. van Vliet, and G. J. Woeginger. An Optimal Algorithm for Preemptive On-line Scheduling. *Operations Research Letters*, 18:127–131, 1995.
9. G. Dobson. Scheduling Independent Tasks on Uniform Processors. *SIAM Journal on Computing*, 13(4):705–716, 1984.
10. L. Epstein. Optimal Preemptive On-Line Scheduling on Uniform Processors with Non-Decreasing Speed Ratios. *Operations Research Letters*, 29(2):93–98, 2001. Also in STACS 2001.
11. L. Epstein and L. M. Favrholdt. Optimal non-preemptive semi-online scheduling on two related machines. In *Proc. of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS'2002)*, pages 245–256, 2002.
12. L. Epstein and L. M. Favrholdt. Optimal preemptive semi-online scheduling to minimize makespan on two related machines. *Operations Research Letters*, 30(4):269–275, 2002.

13. L. Epstein, J. Noga, S. S. Seiden, J. Sgall, and G. J. Woeginger. Randomized Online Scheduling on Two Uniform Machines. *Journal of Scheduling*, 4(2):71–92, 2001.
14. L. Epstein and J. Sgall. A Lower Bound for On-Line Scheduling on Uniformly Related Machines. *Operations Research Letters*, 26(1):17–22, 2000.
15. U. Faigle, W. Kern, and G. Turan. On the performance of online algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.
16. R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling*, 3(5):343–353, 2000.
17. D. K. Friesen. Tighter Bounds for LPT Scheduling on Uniform Processors. *SIAM Journal on Computing*, 16(3):554–560, 1987.
18. T. Gonzalez, O. H. Ibarra, and S. Sahni. Bounds for LPT Schedules on Uniform Processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.
19. T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2000)*, pages 564–565, 2000.
20. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
21. R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:416–429, 1969.
22. E. Horwath, E. C. Lam, and R. Sethi. A Level Algorithm for Preemptive Scheduling. *J. Assoc. Comput. Mach.*, 24:32–43, 1977.
23. B. Kalyanasundaram and K. Pruhs. Maximizing job completions online. In *The 6th Annual European Symposium on Algorithms (ESA'98)*, pages 235–246, 1998.
24. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):214–221, 2000.
25. D. Karger, S. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, 1996.
26. T. W. Lam and K. K. To. Trade-offs between speed and processor in hard-deadline scheduling. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, pages 623–632, 1999.
27. P. Mireault, J. B. Orlin, and R. V. Vohra. A Parametric Worst Case Analysis of the LPT Heuristic for Two Uniform Machines. *Operations Research*, 45:116–125, 1997.
28. J. F. Rudin III. *Improved bounds for the on-line scheduling problem.* PhD thesis, The University of Texas at Dallas, May 2001.
29. S. Seiden. Preemptive Multiprocessor Scheduling with Rejection. *Theoretical Computer Science*, 262(1-2):437–458, 2001.
30. S. Seiden, J. Sgall, and G. Woeginger. Semi-online scheduling with decreasing job sizes. *Operations Research Letters*, 27(5):215–221, 2000.
31. J. Sgall. A Lower Bound for Randomized On-Line Multiprocessor Scheduling. *Inf. Process. Lett.*, 63(1):51–55, 1997.
32. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.
33. J. Wen and D. Du. Preemptive On-Line Scheduling for Two Uniform Processors. *Operations Research Letters*, 23:113–116, 1998.

Scheduling AND/OR-Networks on Identical Parallel Machines

Thomas Erlebach[1], Vanessa Kääb[2,⋆], and Rolf H. Möhring[3,⋆⋆]

[1] ETH Zürich, Computer Engineering and Networks Laboratory (TIK)
Gloriastrasse 35, 8092 Zürich, Switzerland
erlebach@tik.ee.ethz.ch
http://www.tik.ee.ethz.ch/~erlebach
[2] Technische Universität Berlin, Fakultät II, Institut für Mathematik, MA 6-1
Straße des 17. Juni 136, 10623 Berlin, Germany
{kaeaeb,moehring}@math.tu-berlin.de,
http://www.math.tu-berlin.de/coga

Abstract. Scheduling precedence constrained jobs on identical parallel machines is a well investigated problem with many applications. AND/OR-networks constitute a useful generalization of standard precedence constraints where certain jobs can be executed as soon as at least one of their direct predecessors is completed. For the problem of scheduling AND/OR-networks on parallel machines, we present a 2-approximation algorithm for the objective of minimizing the makespan. The main idea of the algorithm is to transform the AND/OR constraints into standard constraints. For the objective of minimizing the total weighted completion time on one machine, scheduling AND/OR-networks is as hard to approximate as LABEL COVER. We show that list scheduling with shortest processing time rule is an $O(\sqrt{n})$-approximation for unit weights on one machine and an n-approximation for arbitrary weights.

1 Introduction

Scheduling precedence constrained jobs on identical parallel machines is a well investigated problem with many applications. A precedence constraint of the form (i, j) means that job j can be started only after the completion of job i. If there are several jobs i such that (i, j) is a precedence constraint, the job j can be started only after *all* of these jobs i are completed (AND-constraint). A natural and useful generalization of standard precedence constraints are AND/OR precedence constraints, represented by AND/OR-networks. In addition to standard constraints, they allow to specify that a node can begin execution as soon as *at least one* of its direct predecessors is completed (OR-constraint). AND/OR-networks arise in many applications, such as resource-constrained project scheduling [18] and assembly/disassembly sequencing [9]. In the latter application, a given product may have to be disassembled. Certain components can be removed only

⋆ This work was accomplished as a member of the European Graduate Program 'Combinatorics, Geometry, and Computation', supported by the Deutsche Forschungsgemeinschaft (DFG) under grant GRK 588/2.

⋆⋆ Supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Mo 446/3-4.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 123–136, 2004.
© Springer-Verlag Berlin Heidelberg 2004

after the removal of other components, leading to standard AND-constraints. However, it might also be possible to remove a component from one of several geometric directions, assuming that other components blocking *this direction* have been removed beforehand. This case naturally leads to OR-constraints. Therefore, the different possibilities how to reach a component can be suitably modeled by AND/OR precedence constraints.

If a given set of jobs with AND/OR precedence constraints has to be executed on a bounded number of identical machines (e.g., assembly lines, workers, etc.), interesting optimization problems arise. Natural objectives are the makespan (the completion time of the job that finishes last), the total completion time (the sum of the completion times of all jobs), and the total weighted completion time (where the completion time of each job is multiplied with the weight of the job). While these problems have been studied intensively for standard precedence constraints, only little is known about scheduling AND/OR-networks on one or several machines.

1.1 Known Results

Scheduling with the makespan objective is trivial on one machine if the jobs are not restricted at all, but also if standard or AND/OR precedence constraints are imposed among the jobs. The problem on identical parallel machines is NP-complete even without precedence constraints and with just two machines. Nevertheless, Graham's list scheduling provides a simple 2-approximation for scheduling precedence constrained jobs on any number of parallel machines [10]. Gillies and Liu [7] present a 2-approximation for scheduling jobs of an acyclic AND/OR-network. Their algorithm first transforms the AND/OR-network into a standard precedence graph and then applies Graham's list scheduling.

Minimizing the total weighted completion time is a non-trivial problem even on a single machine. If there are no precedence constraints among the jobs, it is well known that scheduling the jobs according to *Smith's rule* [20] (in order of non-decreasing processing time over weight ratio) yields an optimal solution. For the total weighted completion time of standard precedence constrained jobs on one machine, 2-approximation algorithms have been obtained with various techniques [12, 3, 2, 17]. For the problem with identical parallel machines, a 4-approximation algorithm was presented in [19]. Unfortunately, it seems that the techniques used to obtain these algorithms resist any application to AND/OR precedence constraints.

1.2 Our Results

We study scheduling problems with AND/OR precedence constrained jobs on one or several machines. For the problem of minimizing the makespan on identical parallel machines, we extend the algorithm of Gillies and Liu [7] to general feasible AND/OR-networks (which may contain cycles) in order to obtain a 2-approximation algorithm. Then we consider the problem of minimizing the total weighted completion time on a single machine. For the special case with unit weights (i.e., total completion time), we prove that list scheduling with *shortest processing time rule* is an $O(\sqrt{n})$-approximation, where n is the number of jobs. This bound is tight. The case with ar-

bitrary weights seems to be harder to approximate. We observe that a reduction from LABEL COVER proposed by Goldwasser and Motwani in [8, 9] together with an improved inapproximability result for LABEL COVER by Dinur and Safra [4] shows that minimizing the total weighted completion time to within a factor of $2^{\log^{1-1/\log\log n} n}$ of the optimum is NP-hard for any $c < 1/2$. We prove that list scheduling with shortest processing time rule is a simple n-approximation for the problem and that this bound is again tight.

2 Preliminaries

2.1 Problem Description

A scheduling instance consists of a set of n *jobs* $V = \{1, 2, \ldots, n\}$. With each job $j \in V$ we associate a (strictly) positive *processing time* p_j and a non-negative *weight* ω_j. In addition to the jobs, a number m of identical parallel *machines* will be given. The machines all run with the same speed and can process any job. Of course, at any point in time, each machine can process only one job and every job can only be processed by one machine. We will restrict to the non-preemptive case. Once a job is started on a certain machine, it will be finished on that machine without any interruption.

The jobs are subject to two different classes of constraints. On one hand, standard *precedence constraints* represented by a directed acyclic graph $G = (V, E(G))$ are given. An edge $(i, j) \in E(G)$ represents the constraint that j has to wait for the completion of i. In a feasible realization of such a problem, all jobs have to be executed in accordance to the partial order induced by G. Each job $j \in V$ has to wait for the completion of all its predecessors in the partial order and thus will also be called an *AND-node*. On the other hand, we allow for precedence relations of the form that a job j has to wait for the completion of only one predecessor. Those restrictions cannot be captured by the classical precedence constraints described above. The model of standard precedence constraints can be generalized by the introduction of a set W of waiting conditions. A *waiting condition* is an ordered pair $w = (X, j)$, where $X \subseteq V$ is a set of jobs and $j \in V \setminus X$ is the *waiting job*. The waiting job j can be processed as soon as one of the jobs in X has been completed. The standard precedence constraints together with the waiting conditions can be represented by an *AND/OR-network* $N = (V \cup W, E)$ in the following way: for every waiting condition $w = (X, j) \in W$, we introduce an *OR-node*, which we will denote by w again. For every $x \in X$, we introduce a directed edge (x, w). In addition, there is a directed edge (w, j) for the waiting job j. To model the required constraints correctly, we impose the rule that an OR-node w can be scheduled as soon as any of its predecessors $x \in X$ is completed. An OR-node $w \in W$ can be considered as a pure dummy node with processing time $p_w = 0$ and weight $\omega_w = 0$. For convenience, we assume N to have a common start vertex, the *source* s, and a common end vertex, the *sink* t, with $p_s = p_t = 0$ and $\omega_s = \omega_t = 0$. For an illustration, an AND/OR-network is depicted in the left part of Fig. 1, where node w_3 represents the waiting condition $(\{5, 6\}, 7)$, for example.

In contrast to standard precedence constraints, an AND/OR-network may contain cycles and be feasible at the same time. Remember that the AND/OR-network $N =$

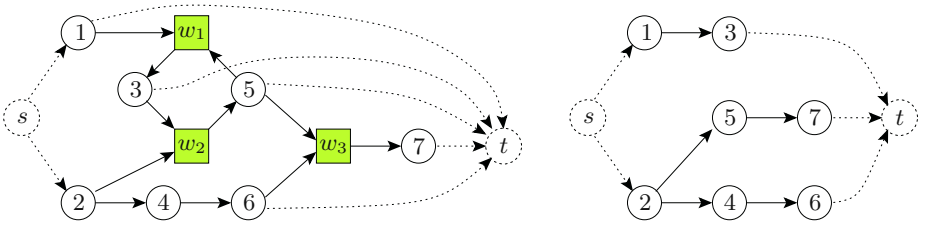


Fig. 1. An AND/OR-network N on the left and a realization of N on the right side. AND-nodes are drawn as circles and OR-nodes as shaded squares.

$(V \cup W, E)$ contains the precedence digraph G as a subgraph. A *realization* of an AND/OR-network N is a partial order $R = (V, <_R)$ which is an extension of G, i.e. $i <_R j$ for each $(i, j) \in E$ with $i, j \in V$, and

$$\text{for each } w = (X, j) \in W, \text{ there exists } x \in X \text{ with } x <_R j.$$

A linear or total order $L = (V, <_L)$, which is a realization of N, is called *linear realization*. In Fig. 1 a realization of the AND/OR-network N on the left is depicted on the right hand side. A total order is given by $L = 1, 2, 3, 4, 5, 6, 7$, for example. An AND/OR-network N is called *feasible* if it has a realization R. We refer to [13, 18] for additional characterizations of feasible AND/OR-networks and a linear time algorithm to check for feasibility.

For a given instance $(N = (V \cup W, E), p, \omega, m)$, a feasible *schedule* is a non-negative vector of start times $S = (S_1, \ldots, S_n)$ for the jobs and waiting conditions such that

1. $S_j \geq \max\{S_v + p_v \mid (v, j) \in E\}$ for all $j \in V$ (*AND-constraints*)
2. $S_w \geq \min\{S_v + p_v \mid (v, w) \in E\}$ for all $w \in W$ (*OR-constraints*)
3. $|\{j \in V \mid S_j \leq t < S_j + p_j\}| \leq m$ at any time $t \geq 0$ (*machine constraints*)

For a given schedule S, the corresponding completion time vector C is defined by $C_j = S_j + p_j$. The makespan of schedule S is denoted by $C_{\max}(S) = \max_{j \in V} C_j = C_t$. The total completion time of S is $\sum_{j \in V} C_j$ and the total weighted completion time is $\sum_{j \in V} \omega_j C_j$.

Let us consider the problem without machine constraints, i.e. $m = \infty$. By definition, $S = (\infty, \ldots, \infty)$ is a feasible schedule and if $S = (S_1, \ldots, S_n)$ and $S' = (S'_1, \ldots, S'_n)$ are feasible schedules, then also the component wise minimal schedule $S'' = (\min\{S_1, S'_1\}, \ldots, \min\{S_n, S'_n\})$. It follows that there exists a (unique) component wise minimal schedule, the so-called *earliest start schedule* (ES). We assume strictly positive job processing times, thus an earliest start schedule can be computed by a modification of Dijkstra's shortest path algorithm in polynomial time. For further details we refer to [13]. Similar algorithms have been presented before in [14, 1, 18]. The earliest start schedule ES has the following property. For every job $j \in V$, the inequality $ES_j \geq \max\{ES_v + p_v \mid (v, j) \in E\}$ is fulfilled with equality for at least one edge $(v, j) \in E$ as otherwise j could start earlier. For the same reason, for each OR-node $w \in W$, the inequality $ES_w \geq \min\{ES_v + p_v \mid (v, w) \in E\}$ is also fulfilled

with equality for at least one edge. We call those edges (v, j) and (v, w), for which equality holds, *tight*.

In the earliest start schedule, every job is started as early as possible without violating any constraint. Therefore it also finishes as early as possible. For the problem without machine constraints it follows that the earliest start schedule ES is an optimal schedule for all three objective functions, the makespan, the total completion time, and the total weighted completion time. In the following we will always denote an *optimal schedule* for a given problem by $S^* = (S_1^*, \ldots, S_n^*)$.

To describe the problem under consideration quickly we will use the $\alpha|\beta|\gamma$-notation of Graham, Lawler, Lenstra, and Rinnooy Kan [11]. The first entry, α, specifies the machine environment, β describes the job characteristics, where $\beta = ao\text{-}prec$ denotes AND/OR precedence constraints, and γ specifies the objective function. We refer to [11, 16] for a detailed description. As an example, the problem of scheduling jobs with unit processing times and AND/OR precedence constraints on one machine with the objective to minimize the total completion time is specified by $1|ao\text{-}prec, p_j = 1|\sum C_j$.

2.2 List Scheduling

One of the most basic and intuitive approaches to tackle a given scheduling problem is *list scheduling*. The idea is to order the jobs according to some priority rule, which gives an ordered list. The jobs are then scheduled according to this priority list. Whenever a machine is free, the first job in the list that has not been scheduled yet and is available at that time is assigned to be processed by the free machine. List scheduling can easily be implemented to run in polynomial time. According to [16], this algorithm has been presented first by Graham [10] and we will also refer to it as *Graham's list scheduling* (GLS).

In the literature, list scheduling is also called a priority-driven heuristic, according to the priority list that guides the order in which the jobs are scheduled. Priority driven heuristics never intentionally leave machines idle. A machine is only left idle if there are currently no jobs available. This means that for every job in the list, at least one predecessor is still in process. Recursing this argument gives a well known fact for standard precedence constraints stated in the next lemma. Let $G = (V, E)$ be a standard (acyclic) precedence graph with a strictly positive processing time vector p and $u, v \in V$ two nodes such that there exists a u-v-path in G. The length of a longest path from u to v shall be denoted by $\ell_{uv}^{\max}$, where $\ell_{uv}^{\max} = p_v$ if $u = v$ and $\ell_{uv}^{\max} = p_v + \max\{\ell_{ux}^{\max} \mid (x, v) \in E$ and there exists a u-x-path in $G\}$ if $u \neq v$.

Lemma 1 (Graham [10]). *In any list schedule for a set of precedence constrained jobs, there is an s-t-path of jobs that is executed during all periods when some machine is idle, and the length of this path is not longer than the makespan of an optimal schedule.*

The big advantage of Graham's list scheduling is that it is easy, both to understand and to implement, and that it is applicable for a wide range of problems.

3 The Makespan on Identical Parallel Machines

Minimizing the makespan of a set of precedence constrained jobs on one machine is trivial. If the jobs are not restricted at all, we can simply schedule them in any order

Algorithm 1: Earliest Start List Scheduling.

Input: AND/OR-network $N = (V \cup W, E)$, number $m \geq 1$ of machines, processing time vector $p > 0$
Output: schedule $S = (S_1, \ldots, S_n)$

compute earliest start schedule ES for (N, p) (no machine constraints);
let $G = (V, E(G))$ be the subgraph of N induced by V;
foreach *OR-node* $w = (X, j) \in W$ **do**
  for *exactly one* $x \in X$ *with* $ES_x + p_x = ES_w$ **do**
    add edge (x, j) to $E(G)$;
let L be a linear extension of the resulting AND-graph;
apply list scheduling to the AND-graph G with p and L to obtain the schedule S;
return S;

without interruption and idle time in between. If standard precedence constraints are involved, we can process the jobs in order of a linear extension of the partial order G representing the precedences. In the case of AND/OR precedence constraints, the solution is equally simple, the jobs can be executed in the order of a linear realization of the AND/OR-network. In every case, the makespan is equal to the sum of the processing times of the jobs, assuming that the constraints are feasible of course.

Now consider the case of scheduling a set of precedence constrained jobs on m identical parallel machines. The problem $Pm|\,|C_{\max}$ is already NP-complete for $m = 2$, but can be solved in pseudo-polynomial time for any fixed number m of machines. For an arbitrary number of machines, i.e. $P|\,|C_{\max}$, the problem is NP-complete in the strong sense [5].

Nevertheless, Graham's list scheduling performs provably well for this problem. Let S^{GLS} denote a list schedule for $P|\,|C_{\max}$ on m machines. Graham [10] proved that, for any instance, $C_{\max}(S^{\mathrm{GLS}}) \leq (2 - \frac{1}{m})\, C_{\max}(S^*)$. He also showed that this performance guarantee is not affected by precedence constraints. Thus list scheduling is a 2-approximation for $P|prec|C_{\max}$.

As a generalization of standard precedence constraints, the problem of minimizing the makespan of AND/OR constrained jobs on identical parallel machines, $P|ao\text{-}prec|C_{\max}$, is NP-complete in the strong sense. Fortunately, the problem with AND/OR precedence constraints can be reduced to the problem with standard precedence constraints preserving the approximation guarantee. Gillies and Liu [7] present a 2-approximation algorithm for acyclic AND/OR-networks. The basic idea of the so-called *minimum path heuristic* in [7] is to first change the AND/OR-network into an AND-only network, i.e. a standard precedence digraph, and then apply Graham's list scheduling. For this transformation, Gillies and Liu use a recursive argument which minimises the longest path to each OR-node provided that only AND-nodes precede the OR-node. This construction fails in the presence of cycles. We present an algorithm similar to the minimum path heuristic which is applicable to AND/OR-networks containing cycles.

The strategy of our *earliest start list scheduling* (ESL) presented in Algorithm 1 is to fix the predecessor of an OR-node to one of the tight (with respect to the earliest start schedule) direct predecessors. Then the OR-node can be removed by making the cho-

sen predecessor a direct predecessor of the immediate successor of the OR-node. Note that the resulting precedence graph G is a realization of N. In addition, it minimises the longest path to each OR-node, respectively its waiting job, among all realizations. Every step of the earliest start list scheduling can be executed in polynomial time, thus the earliest start list scheduling can be implemented to run in polynomial time. Together with Lemma 1 by Graham and the optimality of the earliest start schedule with respect to the makespan objective without machine constraints, we are able to prove an approximation guarantee of 2.

Theorem 1. *Let S^{ESL} denote an earliest start list schedule and S^* an optimal schedule for an instance of $P|ao\text{-}prec|C_{\max}$. Then*

$$C_{\max}(S^{\mathrm{ESL}}) \leq (2 - \frac{1}{m})\, C_{\max}(S^*).$$

Moreover, this bound is tight.

Proof. Consider an earliest start list schedule S^{ESL} and its makespan $C_{\max}(S^{\mathrm{ESL}})$. At any time $0 < t \leq C_{\max}(S^{\mathrm{ESL}})$ either all machines are busy or one or more machines are idle. Accordingly, we can divide the time between 0 and $C_{\max}(S^{\mathrm{ESL}})$ into busy periods and idle periods. We denote the total length of all busy periods by T_b and the total length of all idle periods by T_i, thus $C_{\max}(S^{\mathrm{ESL}}) = T_b + T_i$.

For the total length of idle periods, $T_i \leq \ell_{st}^{\max}(G)$ by Lemma 1. From the construction of graph G we get that $\ell_{st}^{\max}(G) = C_{\max}(ES)$, where ES is the earliest start schedule of (N, p) without machine constraints computed in the first step of the algorithm. The earliest start schedule achieves the optimal makespan for an instance (N, p) and therefore is a lower bound on $C_{\max}(S^*)$. Together this yields $T_i \leq C_{\max}(S^*)$.

Additionally, we can compare the total processing time of all jobs with the makespan of S^* and S^{ESL}. A trivial lower bound on the makespan of any feasible schedule—and thus also an optimal schedule—is $\frac{1}{m}\sum_{j\in V} p_j$. The earliest start list scheduling has to process all the jobs. During a busy period, all m machines process some job, while during an idle period at least one machine processes some job. We therefore get that $mT_b + 1T_i \leq \sum_{j\in V} p_j \leq mC_{\max}(S^*)$. The worst case for $C_{\max}(S^{\mathrm{ESL}}) = T_b + T_i$ subject to the presented constraints is achieved for $T_i = C_{\max}(S^*)$ and $T_b = (1 - 1/m)C_{\max}(S^*)$, which yields the result.

Examples for standard precedence graphs that achieve the worst-case bound of Graham's list scheduling can be found in [10] or [6]. If we apply the earliest start list scheduling to such instances, there is nothing to be done in the first part of the algorithm and the performance guarantee will be achieved by the last part, the list scheduling. This proves the tightness of the stated approximation ratio and completes the proof. □

We observe that the presence of additional OR constraints does not seem to make it any harder to minimize the makespan of precedence constrained jobs on identical parallel machines. Unfortunately the situation changes completely for the total weighted completion time objective.

4 The Total Weighted Completion Time on One Machine

The problem of minimizing the total weighted completion time on one machine is NP-hard in the strong sense as soon as precedence constraints are involved [15]. Again this complexity result carries over to AND/OR precedence constraints.

In contrast to the makespan objective, minimizing the total weighted completion time of a set of AND/OR constrained jobs is much harder than for standard precedence constraints. In fact, it is not approximable within any reasonable factor. Therefore we can expect that the successful approaches for standard precedence constraints will not be applicable to AND/OR precedence constraints. We will examine the performance of list scheduling with shortest processing time (SPT) rule on the problem $1|ao\text{-}prec|\sum \omega_j C_j$ for both cases of unit and arbitrary weights. The SPT rule simply orders the jobs according to non-decreasing processing times. It turns out that the greedy approach of list scheduling with SPT rule results in a useful property of the computed schedule with respect to any other feasible schedule.

Let $(N = (V \cup W, E), p, \omega)$ be an instance of $1|ao\text{-}prec|\sum \omega_j C_j$ and let S^{SPT} with completion time vector C^{SPT} be the schedule computed for this instance by list scheduling with SPT rule. Consider an arbitrary feasible schedule S and its completion time vector C for (N, p, ω). For any job $j \in V$ we define a *threshold* $\xi^j(S)$ with respect to schedule S: $\xi^j(S) = \max\{p_i \mid C_i \leq C_j\}$, i.e. the maximum processing time of a job equal to j or scheduled before j in S. The vector of thresholds, $\xi(S) = (\xi^1(S), \ldots, \xi^n(S))$, is called the threshold of schedule S.

Lemma 2. *Let S be a feasible schedule for an instance (N, p, ω) with threshold $\xi(S)$. Then for every job $j \in V$ it holds that $p_i \leq \xi^j(S)$ for all i with $C_i^{\mathrm{SPT}} \leq C_j^{\mathrm{SPT}}$.*

Proof. The proof is accomplished by induction along the order of the jobs in the feasible schedule S. To this end we assume without loss of generality that the jobs are numbered such that $S_1 < S_2 < \cdots < S_n$. In addition, we write ξ for short instead of $\xi(S)$.

First we make the following observation for schedule S^{SPT}, which we will refer to as the *greedy choice argument*. Let t be the point in time, at which job j becomes available, i.e. in some realization of N all jobs in the predecessor set of j have been completed before t. Then, by the greedy choice of list scheduling with SPT rule, no job with a longer processing time than j itself (and therefore coming after j in the list) will be scheduled between t and S_j.

Consider job 1. By definition, $\xi^1 = p_1$. In addition, job 1 is available at time $t = 0$ as it is scheduled at that time in the feasible schedule S and thus cannot have any direct predecessors except s. The claim follows by the greedy choice argument.

Now consider job k and assume that for all $j = 1, \ldots, k-1$ it holds that $p_i \leq \xi^j$ for all i with $C_i^{\mathrm{SPT}} \leq C_j^{\mathrm{SPT}}$. We have to distinguish between the two cases that the threshold of k is greater or equal to the threshold of $k-1$.
Case a) $\xi^k > \xi^{k-1}$. From the definition of ξ it follows that $\xi^k = p_k$ and thus $p_k > \xi^{k-1}$. Let $x \in \{1, \ldots, k-1\}$ be the job that finishes last in the list schedule, i.e. $C_x^{\mathrm{SPT}} = \max\{C_j^{\mathrm{SPT}} \mid j = 1, \ldots, k-1\}$. By assumption, $p_i \leq \xi^x \leq \xi^{k-1} < p_k$ for all i with $C_i^{\mathrm{SPT}} \leq C_x^{\mathrm{SPT}}$. We conclude that k is scheduled after x in the list schedule and $p_i \leq \xi^k$ for all i with $C_i^{\mathrm{SPT}} \leq C_x^{\mathrm{SPT}}$. From the feasibility of schedule S and the maximal choice

of x in S^{SPT} it follows that k is available at time C_x^{SPT}. By the greedy choice argument, no job with a strictly longer processing time than k will be scheduled between C_x^{SPT} and C_k^{SPT}, which proves the first case.
Case b) $\xi^k = \xi^{k-1}$. In this case, k may finish before or after some other job $j < k$. Again let $x \in \{1, \ldots, k-1\}$ be such that $C_x^{\mathrm{SPT}} = \max\{C_j^{\mathrm{SPT}} \mid j = 1, \ldots, k-1\}$. If k completes before x, i.e. $C_k^{\mathrm{SPT}} < C_x^{\mathrm{SPT}}$ then the claim trivially holds as by assumption $p_i \leq \xi^x \leq \xi^{k-1} = \xi^k$ for all i with $C_i^{\mathrm{SPT}} \leq C_x^{\mathrm{SPT}}$. If on the other hand k completes after x, we can again argue as in case a). For the jobs completed before C_x^{SPT} the claim holds by assumption and for the jobs completed between C_x^{SPT} and C_k^{SPT} it follows by the greedy choice argument. □

Note that this property of S^{SPT} holds for the threshold of any feasible schedule S and thus in particular for the threshold of an optimal solution S^*, independent of the optimality criterion.

Theorem 2. *Scheduling a set of n AND/OR precedence constrained jobs in order of non-decreasing processing times (SPT) is an $O(\sqrt{n})$-approximation for the problem $1|ao\text{-}prec|\sum C_j$. Moreover, this bound is tight.*

Proof. Consider the schedule S^{SPT} with completion time vector C^{SPT} for an instance (N, p) of $1|ao\text{-}prec|\sum C_j$ with n jobs. Let $x \in V$ be the last job in the list schedule for which x itself and every job completed before x in the list schedule have a processing time smaller than or equal to $C_x^{\mathrm{SPT}}/\sqrt{n}$, if such a job exists. Formally, x is chosen such that $C_x^{\mathrm{SPT}} = \max_{j \in V}\{C_j^{\mathrm{SPT}} \mid p_i \leq C_j^{\mathrm{SPT}}/\sqrt{n}\ \forall i \text{ with } C_i^{\mathrm{SPT}} \leq C_j^{\mathrm{SPT}}\}$. If such an x exists, let $V^{\leq} = \{j \in V \mid C_j^{\mathrm{SPT}} \leq C_x^{\mathrm{SPT}}\}$ denote the set of jobs that are scheduled before x including x itself. The set of jobs scheduled after x in the list schedule shall be denoted by $V^{>} = \{j \in V \mid C_j^{\mathrm{SPT}} > C_x^{\mathrm{SPT}}\}$. If no such x exists, then $V^{\leq} = \emptyset$ and $V^{>} = V$. We have to treat the jobs in the two sets separately.

First consider $V^{\leq}$. The total completion time of the list schedule for the jobs in $V^{\leq}$ can be generously bounded from above by $\sum_{j \in V^{\leq}} C_j^{\mathrm{SPT}} \leq nC_x^{\mathrm{SPT}}$. Now we have to bound the total completion time of an optimal schedule for the jobs in $V^{\leq}$ from below. By construction, $p_j \leq C_x^{\mathrm{SPT}}/\sqrt{n}$ for every $j \in V^{\leq}$. Thus, there are at least $r \geq \sqrt{n}$ jobs in $V^{\leq}$ and we want to minimize their total completion time. Let $j_1, \ldots, j_r$ be the jobs in $V^{\leq}$ such that $C_{j_1}^* < \cdots < C_j^*$. Since $\sum_{i=1}^{r} p_j \geq C_x^{\mathrm{SPT}}$, we obtain the following inequality: $\sum_{j \in V^{\leq}} C_j^* \geq C_x^{\mathrm{SPT}} + (C_x^{\mathrm{SPT}} - p_j) + (C_x^{\mathrm{SPT}} - p_j - p_{j\ -1}) + \cdots + p_1$. The sum on the right hand side is minimized if we descend from the maximal summand C_x^{SPT} as fast as possible and in as few steps as possible. Since $p_j \leq C_x^{\mathrm{SPT}}/\sqrt{n}$ for $i = 1, \ldots, r$, this is achieved if the processing time of every job j_i, $i = 1, \ldots, r$, takes its maximum possible value $p_j = C_x^{\mathrm{SPT}}/\sqrt{n}$. We then get $\sum_{j \in V^{\leq}} C_j^* \geq \sum_{i=1}^{\sqrt{n}} iC_x^{\mathrm{SPT}}/\sqrt{n} \geq (\sqrt{n}/2)C_x^{\mathrm{SPT}}$. Combining upper and lower bound for the jobs in $V^{\leq}$ yields

$$\sum_{j \in V^{\leq}} C_j^{\mathrm{SPT}} \leq 2\sqrt{n} \sum_{j \in V^{\leq}} C_j^* \ . \tag{1}$$

Now consider the jobs in $V^{>}$. By definition, for every $j \in V^{>}$ there exists a job k with $p_k > C_j^{\mathrm{SPT}}/\sqrt{n}$ that is either equal to j or scheduled before j in the list schedule.

Consider an optimal schedule S^* and its corresponding threshold $\xi^j(S^*)$ for j. By Lemma 2 we know that j and every job scheduled before j in the list schedule has a processing time smaller than or equal to $\xi^j(S^*)$. This yields

$$\frac{C_j^{\mathrm{SPT}}}{\sqrt{n}} \;<\; p_k \;\leq\; \xi^j(S^*) \;\leq\; C_j^* \;\;, \tag{2}$$

as $\xi^j(S^*)$ is a trivial lower bound on C_j^* by definition.

Combining (1) and (2) proves the approximation guarantee:

$$\sum_{j\in V} C_j^{\mathrm{SPT}} \leq \sum_{j\in V^{\leq}} C_j^{\mathrm{SPT}} + \sum_{j\in V^{>}} C_j^{\mathrm{SPT}} \leq 2\sqrt{n} \sum_{j\in V^{\leq}} C_j^* + \sum_{j\in V^{>}} \sqrt{n} C_j^* \leq 2\sqrt{n} \sum_{j\in V} C_j^* \;\;.$$

We have shown that list scheduling with SPT rule is a ρ-approximation algorithm with $\rho \in O(\sqrt{n})$, it remains to prove that also $\rho \in \Omega(\sqrt{n})$. Already for standard precedence graphs, the approximation ratio for list scheduling with SPT rule is bounded from below by $\Omega(\sqrt{n})$ and therefore this also holds for the general case of AND/OR precedence constraints. We will demonstrate this by presenting a series of digraphs that force the solution obtained by list scheduling with SPT rule to be a factor of $\Theta(\sqrt{n})$ away from the optimum.

Let $k > 0$ be an integer and consider the following precedence graph $G = (V, E)$. The set of jobs V consists of k jobs $i_1, \ldots, i_k$, one job x, and k^2 jobs $j_1, \ldots, j_{k^2}$ that form a chain, preceded by x. Thus in E there are the edges (x, j_1) and $(j_\kappa, j_{\kappa+1})$ for all $1 \leq \kappa \leq k^2 - 1$. We have the following job processing times: $p_i = k^2$, for $\kappa = 1, \ldots, k$, $p_x = k^2$, and $p_j = 1$, for all $\kappa = 1, \ldots, k^2$.

The SPT rule orders the jobs according to non-decreasing processing time and thus a possible ordering is $L = j_1, \ldots, j_{k^2}, i_1, \ldots, i_k, x$. Note that we can force the SPT rule to construct this bad list L by slightly modifying the processing times. List scheduling then computes a schedule with the following order of the jobs: $i_1, \ldots, i_k, x, j_1, \ldots, j_{k^2}$. For the objective value of this schedule we get that

$$\sum_{j\in V} C_j^{\mathrm{SPT}} \geq \sum_{\ell=1}^{k+1} \ell k^2 + k^2(k+1)k^2 = \Omega(k^5) \;\;. \tag{3}$$

The optimal schedule prefers job x, which can release the long chain of short jobs. Therefore, an exact algorithm produces the schedule: $x, j_1, \ldots, j_{k^2}, i_1, \ldots, i_k$. For the value of this optimal schedule we can calculate that

$$\sum_{j\in V} C_j^* \leq (k^2+1)2k^2 + \sum_{\ell=3}^{k+2} \ell k^2 = O(k^4) \;\;. \tag{4}$$

Thus the performance ratio of list scheduling for this instance is $\Omega(k^5)/O(k^4) = \Omega(k)$. The precedence graph has $n \in \Theta(k^2)$ many vertices, which yields an approximation guarantee of $\Theta(\sqrt{n})$ for list scheduling with SPT rule on this instance for n respectively k going towards infinity. □

Coming back to the weighted problem $1|ao\text{-}prec|\sum \omega_j C_j$, we are actually able to establish a strong inapproximability result. Goldwasser and Motwani [8] present an approximation preserving reduction from LABEL COVER to a special case of scheduling with AND/OR precedence constraints arising in disassembly problems.

An instance of a problem considered in [8] is given by a set of jobs, called *tasks*, with unit processing times. The jobs are either AND- or OR-nodes and there are no weights involved. The AND/OR-network representing the precedence constraints is assumed to have *internal-tree* structure, which is a slight generalisation of a tree. We call a node of the network a *leaf*, if it has no direct predecessors. An AND/OR-network N has internal-tree structure, if $N \setminus \{j \in V \mid j \text{ is a leaf}\}$ is an in-tree. The scheduling model requires that for an AND-node to be scheduled, all its direct predecessors have to be processed before the AND-node, and for an OR-node only one direct predecessor has to be scheduled before the OR-node and the others may be left completely unprocessed. The goal of AND/OR scheduling is to minimize a) the number of leaves or b) the number of jobs (AND- and OR-nodes) that need to be scheduled to be able to schedule some sink y of the network, and thus solving the problem instance. Goldwasser and Motwani show that LABEL COVER is a special case of scheduling AND/OR constrained jobs subject to objective a) and that a) can be reduced to b), preserving the approximation guarantee.

Minimizing the number of jobs, i.e. objective b), that need to be scheduled to be able to schedule some sink y of an AND/OR-network can be modeled as a special case of $1|ao\text{-}prec|\sum \omega_j C_j$. In the given AND/OR scheduling instance we assign unit processing times to all AND-nodes and zero processing time to the OR-nodes. In addition we set $\omega_y = 1$ and $\omega_v = 0$ for every other node $v \in V \cup W$. A solution S' to the problem of scheduling a minimum number of AND/OR constrained jobs corresponds to a solution to $1|ao\text{-}prec, p_j = 1|\sum \omega_j C_j$ that minimises the total weighted completion time $\sum_{j \in V} \omega_j C_j = C_y$: Simply schedule all jobs contained in S' before y and all other jobs afterwards. The total weighted completion time is equal to the number of AND-nodes of solution S'.

Together with a strengthened inapproximability result for LABEL COVER by Dinur and Safra [4] we can make the following statement.

Theorem 3. *The scheduling problem $1|ao\text{-}prec|\sum \omega_j C_j$ with unit processing times is NP-hard to approximate within a factor of $2^{\log^{1-\gamma} n}$ of the optimum value, where $\gamma = 1/(\log\log n)^c$ for any constant $c < 1/2$ and $n = |V|$ is the number of jobs.*

For a detailed presentation of the proof we refer to [13]. Theorem 3 shows that $1|ao\text{-}prec|\sum \omega_j C_j$ is a very hard problem even if all processing times are equal to one. Interestingly enough, the next theorem proves that list scheduling with shortest processing time rule is an easy n-approximation for this problem, and it seems difficult to achieve an asymptotically better ratio.

Theorem 4. *Scheduling a set of n weighted AND/OR precedence constrained jobs in order of non-decreasing processing times (SPT) is an n-approximation for the problem $1|ao\text{-}prec|\sum \omega_j C_j$. Moreover, this bound is tight.*

Proof. Let $(N = (V \cup W, E), p, \omega)$ be an instance of $1|ao\text{-}prec|\sum \omega_j C_j$. Without loss of generality assume that the jobs are numbered such that $p_1 \leq p_2 \leq \cdots \leq p_n$.

Let S^{SPT} and C^{SPT} denote the schedule and its completion time vector produced by Graham's list scheduling with the list $L = 1, \ldots, n$. The optimal schedule and its completion time vector are denoted by S^* and C^* as usual. By definition, the threshold $\xi^j(S^*)$ of job j with respect to S^* is a lower bound on C_j^*. Now we have to bound C_j^{SPT} from above. Lemma 2 states that for each $j \in V$ it holds that $p_i \leq \xi^j(S^*)$ for all jobs i with $C_i^{\mathrm{SPT}} \leq C_j^{\mathrm{SPT}}$. For every $j \in V$ we therefore get that

$$C_j^{\mathrm{SPT}} \leq \sum_{p \leq \xi (S^*)} p_i \leq \sum_{p \leq \xi (S^*)} \xi^j(S^*) \leq n\xi^j(S^*) \leq nC_j^* \ ,$$

which yields the upper bound on the approximation ratio on a per job basis.

To prove the lower bound we present a series of AND/OR-networks that force the solution of list scheduling to be a factor of n away from the optimum.

Let $k > 0$ be an integer and consider the following AND/OR-network $N = (V \cup W, E)$. The set of jobs V consists of $k+2$ jobs, $i_1, \ldots, i_k$, x, and j. In addition there is one OR-node $w \in W$. The edge set E contains the edges $(i_\kappa, i_{\kappa+1})$ for $\kappa = 1, \ldots, k-1$, (i_k, w), (x, w), and (w, j). The processing times of all jobs are equal to one. The weights are $\omega_i = 0$ for all $\kappa = 1, \ldots, k$, $\omega_x = 0$, and $\omega_j = 1$.

The constructed AND/OR-network is very simple, it consists of one OR-node w with its direct successor j. The two predecessors of the OR-node w are one chain of k nodes and one single node x. If we sort the jobs in order of non-decreasing processing times, we may get the bad list $L = j, i_1, \ldots, i_k, x$. Again, by slightly modifying the processing times we can force the shortest processing time rule to choose this bad list L. List scheduling then computes the schedule $i_1, \ldots, i_k, j, x$ with an objective function value of $\sum_{j \in V} \omega_j C_j^{\mathrm{SPT}} = k + 1$.

The optimal schedule prefers job x, as it can release j, which is the only job with positive weight. Therefore, an exact algorithm produces the schedule $x, j, i_1, \ldots, i_k$ with a value of $\sum_{j \in V} \omega_j C_j^* = 2$.

The performance ratio of list scheduling for this instance is $\frac{k+1}{2} = \Theta(k)$. The AND/OR-network has $n = k + 2 \in \Theta(k)$ many jobs, which yields a performance ratio of $\Theta(n)$ for the list scheduling algorithm on this instance for n respectively k going towards infinity. □

The constructed example has very restricted values, thus the performance guarantee even holds for the problem $1|ao\text{-}prec, p_j = 1, \omega_j \in \{0, 1\}| \sum \omega_j C_j$. In addition, it also provides a lower bound of $\Omega(n)$ for list scheduling with Smith's rule.

We remark that the ideas from the proof of Theorem 4 can easily be generalized to show that list scheduling with SPT rule is also an n-approximation algorithm for $P|ao\text{-}prec| \sum \omega_j C_j$.

5 Conclusions

We want to briefly summarize the stated results. In Sect. 3 we have seen that minimizing the makespan on identical parallel machines subject to AND/OR precedence constraints can be approximated within a factor of 2. This approximation guarantee was

established by transforming the AND/OR precedence constraints into standard precedence constraints and applying list scheduling to the resulting instance. It is remarkable that in the case of the makespan objective, additional OR constraints do not seem to make the scheduling problem any harder.

In contrast to this, a major gap occurs in the approximability of minimizing the total weighted completion time for standard and AND/OR precedence constrained jobs on a single machine. While the problem can be approximated within a constant factor for standard constraints, the inapproximability result stated in Theorem 3 shows that this is not the case for AND/OR constraints unless P=NP. Therefore, it is not possible to transform the AND/OR constraints into standard precedence constraints such that the approximation ratio for the problem is preserved. In addition, there seems to be a difference in the approximability of the weighted and the unweighted case for AND/OR precedence constraints. For standard precedence constrained jobs, Woeginger proves in [21] that the best possible approximation ratio for minimizing the total weighted completion time is the same for the general problem as well as for a number of special cases including unit processing times, unit weights, and combinations thereof. For AND/OR precedence constrained jobs, the inapproximability result of Theorem 3 and the $O(\sqrt{n})$-approximation algorithm for the unweighted case of Theorem 2 suggest that a gap may exist between the hardness of approximating the total weighted and the total unweighted completion time. We also could not prove any inapproximability result for $1|ao\text{-}prec|\sum C_j$. It is still possible that there exists a constant or at least logarithmic approximation algorithm for the problem $1|ao\text{-}prec|\sum C_j$.

References

1. Adelson-Velsky, G.M., Levner, E.: Project scheduling in AND/OR graphs: A generalization of Dijkstra's algorithm. Technical report, Department of Computer Science, Holon Academic Institute of Technology, Holon, Israel (1999)
2. Chekuri, C., Motwani, R.: Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics* **98** (1999) 29–38
3. Chudak, F., Hochbaum, D.S.: A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters* **25** (1999) 199–204
4. Dinur, I., Safra, S.: On the hardness of approximating label-cover. Electronic Colloquium on Computational Complexity (ECCC) Technical Report TR99-015, School of Mathematical Sciences, Tel Aviv University (1999)
5. Garey, M.J., Johnson, D.S.: *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. Freemann, New York (1979)
6. Gillies, D.W., Liu, J.W.S.: Greed in recource scheduling. *Acta Informatica* **28** (1991) 755–775
7. Gillies, D.W., Liu, J.W.S.: Scheduling tasks with AND/OR precedence constraints. *SIAM Journal on Computing* **24** (1995) 797–810
8. Goldwasser, M.H., Motwani, R.: Intractability of assembly sequencing: Unit disks in the plane. In: *Algorithms and Data Structures*. Volume 1272 of LNCS, Springer (1997) 307–320 Proceedings of the 5th Annual Workshop on Algorithms and Data Structures (WADS'97).
9. Goldwasser, M.H., Motwani, R.: Complexity measures for assembly sequences. *International Journal of Computational Geometry & Applications* **9** (1999) 371–417

10. Graham, R.L.: Bounds for certain multiprocessing anomalies. *Bell System Technical Journal* **45** (1966) 1563–1581
11. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* **5** (1979) 287–326
12. Hall, L.A., Schulz, A.S., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research* (1997) 513–549
13. Kääb, V.: *Scheduling with AND/OR-Networks*. PhD thesis, Technische Universität Berlin, Germany (2003)
14. Knuth, D.E.: A generalization of Dijkstra's algorithm. *Information Processing Letters* **6** (1977) 1–5
15. Lawler, E.L.: Sequencing jobs to minimize total weighted completion time. *Annals of Discrete Mathematics* **2** (1978) 75–90
16. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and scheduling: Algorithms and complexity. In: *Logistics of Production and Inventory*. Volume 4 of Handbooks in Operations Research and Management Science, North-Holland, Amsterdam (1993) 445–522
17. Margot, F., Queyranne, M., Wang, Y.: Decompositions, network flows, and a precedence constrained single machine scheduling problem. Technical Report 2000-29, Department of Mathematics, University of Kentucky, Lexington (2000)
18. Möhring, R.H., Skutella, M., Stork, F.: Scheduling with AND/OR precedence constraints. Technical Report 689/2000, Technische Universität Berlin, Department of Mathematics, Germany (2000) To appear in SIAM Journal on Computing.
19. Munier, A., Queyranne, M., Schulz, A.S.: Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In: *Integer Programming and Combinatorial Optimization*. Volume 1412 of LNCS, Springer (1998) 367–383 Proceedings of the 6th International Conference on Integer Programming and Combinatorial Optimization (IPCO).
20. Smith, W.E.: Various optimizers for single-stage production. *Naval Research Logistics Quarterly* **3** (1956) 59–66
21. Woeginger, G.J.: On the approximability of average completion time scheduling under precedence constraints. In: *Automata, Languages and Programming*. Volume 2076 of LNCS, Springer (2001) 887–897 Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01).

Combinatorial Interpretations of Dual Fitting and Primal Fitting

Ari Freund[1] and Dror Rawitz[2]

[1] Cesarea Rothschild Institute, University of Haifa, Haifa 31905, Israel
arief@cs.haifa.ac.il
[2] Department of Computer Science, Technion, Haifa 32000, Israel
rawitz@cs.technion.ac.il

Abstract. We present two new combinatorial approximation frameworks that are not based on LP-duality, or even on linear programming. Instead, they are based on weight manipulation in the spirit of the *local ratio technique*. We show that the first framework is equivalent to the LP based method of *dual fitting* and that the second framework is equivalent an LP-based method which we define and call *primal fitting*.
Our equivalence results are not unlike the recently proven equivalence between the (combinatorial) local ratio technique and the (LP based) primal-dual schema. Although equivalent, the local ratio approach is more intuitive, and indeed, several breakthrough results were first obtained within the local ratio framework and only later translated into primal-dual algorithms. We believe that our frameworks may have a similar role with respect to their LP-based counterparts (dual fitting and primal fitting). Also, we think that the notion of primal fitting is of independent interest.
We demonstrate the frameworks by presenting alternative analyses to the greedy algorithm for the *set cover* problem, and to known algorithms for the *metric uncapacitated facility location* problem. Also, we analyze a 9-approximation algorithm for a *disk cover* problem that arises in the design phase of cellular telephone networks.

1 Introduction

This paper is motivated by two prominent papers dealing with the *metric uncapacitated facility location* (MUFL) problem. Both papers provide algorithms based on linear programming (LP) duality that deviate from the standard *primal dual* approach. In the first, Jain and Vazirani [16] present a 3-approximation algorithm for MUFL which deviates from the standard primal-dual paradigm in that it does not employ the usual mechanism of relaxing the dual complementary slackness conditions, but rather it relaxes the *primal* conditions. In the second paper Jain et al. [15] develop two *dual fitting* algorithms for MUFL, with approximation factors of 1.861 and 1.61. Dual fitting algorithms produce a feasible primal solution and an *infeasible* dual solution such that: (1) the cost of the dual solution dominates the cost of the primal solution, and (2) dividing the dual solution by an appropriately chosen r results in a feasible dual solution. These two properties imply that the primal solution is r-approximate. This contrasts with

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 137–150, 2004.
© Springer-Verlag Berlin Heidelberg 2004

the standard primal-dual approach, in which a feasible dual solution is found and used to direct the construction of a primal solution. The dual fitting method was (implicitly) used previously to analyze the greedy H_n-approximation algorithm for *set cover* [17, 11]. (H_n is the nth harmonic number, i.e., $H_n = \sum_{i=1}^{n} \frac{1}{i}$.)

MUFL and other facility location problems have received a lot of attention from the approximation community recently. A survey by Shmoys [19] describes some of the techniques used to approximate MUFL and the *k-median* problem. The best approximation factor for MUFL to date is 1.52 [18]. A lower bound (assuming NP $\not\subseteq$ DTIME($n^{O(\log \log n)}$)) of 1.463 is due to Guha and Khuller [13].

In this paper we present two combinatorial approximation frameworks that are not based on LP-duality, or even on linear programming. Rather, they are based on weight manipulation in the spirit of the *local ratio technique* [5, 4]. Both frameworks use the following idea. Two weight functions w_1 and w_2 are used, where $w_2 = r \cdot w_1$, and the weight with respect to w_1 of the solution found is shown to be less than or equal to the optimum with respect to w_2.

In the remainder of this section we briefly describe the primal-dual schema and the dual fitting approach. In Section 2 we present our first framework and show that it is equivalent to dual fitting. We demonstrate this framework by providing alternative analyses of the greedy algorithm for *set cover* and the Jain et al. [15] 1.861-approximation algorithm for MUFL. In Section 3 we present our second framework and define a method called *primal fitting*, which we show is tantamount to it. We demonstrate our second framework and the notion of primal fitting by analyzing two algorithms that where originally designed within the primal-dual paradigm: the Jain-Vazirani [16] 3-approximation algorithm for MUFL, and a 9-approximation algorithm of Chuzhoy [10] for a *disk cover* problem arising in the design phase of cellular telephone networks.

Our equivalence results are not unlike the recently proven [6] equivalence between the (combinatorial) local ratio technique and the (LP based) primal-dual schema. Though equivalent, the local ratio approach is more intuitive—indeed, several breakthrough results such as the first constant factor approximation algorithm for *feedback vertex set* [1, 7, 9] and the first local ratio/primal-dual algorithms for maximization problems [2, 3] were first obtained within the local ratio framework and only later interpreted as primal-dual algorithms. We believe that our frameworks may have a similar role with respect to their LP-based counterparts. Also, we think that the notion of primal fitting is of independent interest.

1.1 Primal-Dual Algorithms

Consider the following linear program and its dual:

$$\begin{array}{lll} \min & \sum_{j=1}^{n} w_j x_j & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \forall i \in \{1, \ldots, m\} \\ & x_j \geq 0 & \forall j \in \{1, \ldots, n\} \end{array} \qquad \begin{array}{lll} \max & \sum_{i=1}^{n} b_i y_i & \\ \text{s.t.} & \sum_{i=1}^{n} a_{ij} y_j \leq w_j & \forall j \in \{1, \ldots, n\} \\ & y_i \geq 0 & \forall i \in \{1, \ldots, m\} \end{array}$$

A primal-dual r-approximation algorithm is an algorithm that constructs an integral primal solution x and (possibly implicitly) a dual solution y, such that x and y satisfy

$$wx \leq r \cdot by. \tag{1}$$

It follows, by weak duality, that x is r-approximate. One way to find such a pair of primal and dual solutions is to focus on pairs (x, y) satisfying the following *relaxed complementary slackness* conditions (for appropriately chosen r_1 and r_2):

$$\begin{array}{ll} \text{Relaxed Primal:} & \forall j,\ x_j > 0 \Rightarrow w_j/r_1 \leq \sum_{i=1}^{m} a_{ij} y_i \leq w_j. \\ \text{Relaxed Dual:} & \forall i,\ y_i > 0 \Rightarrow b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq r_2 \cdot b_i. \end{array}$$

These conditions imply that x is $(r_1 \cdot r_2)$-approximate since

$$\sum_{j=1}^{n} w_j x_j \leq \sum_{j=1}^{n} r_1 \cdot \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j = r_1 \cdot \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i \leq r_1 \cdot r_2 \cdot \sum_{i=1}^{m} b_i y_i.$$

In most primal-dual algorithms only the dual complementary slackness conditions are relaxed, i.e., $r_1 = 1$. (See, e.g., [12, 8].) Typically, such an algorithm begins with the infeasible primal solution $x = 0$ and the feasible dual solution $y = 0$. It then iteratively increases dual variables and "improves" the feasibility of the primal solution, ending with a primal/dual pair satisfying the relaxed dual conditions. (Sometimes a "cleanup" phase is also needed). This design method is commonly referred to as the *primal-dual schema*.

1.2 Dual Fitting

The *dual fitting* method is a variation on the primal-dual theme. The idea is to construct a primal/dual solution pair, but this time the dual solution is *infeasible* and its cost dominates the cost of the primal solution. The key point is that the dual solution has the property that dividing it by r yields a feasible solution. Thus, this *shrunk dual* and the primal solution satisfy Inequality (1), and hence the primal solution is r-approximate. The analysis of a dual fitting algorithm consists therefore of two parts: (1) showing that the dual solution dominates the primal solution (cost-wise), and (2) showing that dividing the dual solution by r makes it feasible, or rather, finding the smallest r for which this is guaranteed.

The dual fitting method was introduced as a design technique by Jain et al. [15]. They recounted Chvatal's work on *set cover* [11] and showed how the factor H_n can be obtained as the solution to the optimization problem mentioned at the end of the previous paragraph. Following that they described two dual fitting algorithms for MUFL achieving approximation factors of 1.861 and 1.61.

2 A Combinatorial Framework Equivalent to Dual Fitting

In this section we show that the dual fitting approach can be described in purely combinatorial terms without recourse to linear programming. We demonstrate this on Chvatal's *set cover* algorithm in full detail, and discuss the principles of our approach at some length. We then sketch briefly how the same can be done for the Jain et al. algorithms for MUFL [15].

2.1 Set Cover

In the *set cover* problem we are given a collection of non-empty sets $\mathcal{C} = \{S_1, \ldots, S_m\}$ and a non-negative weight function w on the sets. Set S_i is said to *cover* element u if $u \in S_i$. The goal is to find a minimum-weight collection of sets that together cover all elements in the universe $U = \bigcup_{i=1}^{m} S_i$.

Recall that the greedy algorithm for set cover selects in each iteration the set S_i minimizing the ratio between its cost and the number of elements covered by it that have not been covered in previous iterations. Algorithm **SC** (appearing below) is our reformulation of the greedy algorithm. To explain it, we find it convenient to imagine sets being paid by elements in order to cover them. The method of payment is that each element issues checks for various amounts to the sets containing it, but these checks are not immediately cashed. Rather, a set cashes its checks (and adds itself to the solution) only when the total sum of these checks is high enough to cover the set's cost. Also, elements occasionally retract some of the (as yet uncashed) checks they have issued.

More specifically, the algorithm proceeds as follows. At the beginning of each iteration a value $\epsilon \geq 0$ is determined, and every uncovered element issues a check for the sum of ϵ to every set containing it. If as a result a set S_i becomes fully funded by the checks in its possession, it cashes these checks and adds itself to the solution. (If several sets become fully funded, one is selected arbitrarily.) Every *newly* covered element then retracts all of the checks it has given to other sets. This process continues until all elements get covered. Intuitively, the motivation for retracting checks is that each element is willing to pay to get covered, and when several elements get covered by a given set, they expect to split the set's cost among them evenly. However, a given element does not know at the outset which set will eventually cover it, so it distributes (equal sum) checks to all of the sets containing it. When the element finally gets covered it retracts the uncashed checks it has issued, since it has no desire to help cover other elements. The outcome is that every element eventually contributes towards the cost of exactly one set—the first set containing it that is added to the solution—and these contributions add up to the total cost of the solution.

We now relate the above description of the algorithm to the pseudo-code below. The algorithm keeps track of the elements already covered by deleting them from all sets in which they are members and discarding sets that become empty. The algorithm keeps track of the checks issued to a set by adjusting the set's weight: when a check is received by a set, the set's weight is decreased by the amount written on it, and when the check is retracted, the set's weight is increased by the same amount. The algorithm keeps track of the total sum of uncashed checks issued by an uncovered element to a set containing it by means of a variable Δ. Since at every moment this amount is simply the sum of ϵs used up to that moment, a single variable suffices for all elements. Finally, in anticipation of the analysis, the algorithm conceptually creates and manipulates (in Lines 2 and 7) a new weight function $w^{\$}$ which is initially set to $r \cdot w$, for some r to be determined later. We stress that this is not really part of the algorithm, and is only included for purposes of the analysis.

Algorithm SC

1. $\Delta \leftarrow 0$.
2. Conceptually: $w^{\$} \leftarrow r \cdot w$.
3. While the problem instance is not empty do:
4. $\quad \epsilon \leftarrow \min_i \{w(S_i)/|S_i|\}$.
5. $\quad$ Define a weight function δ by: $\forall i,\ \delta(S_i) = \epsilon \cdot |S_i|$.
6. $\quad w \leftarrow w - \delta$. (Note that by the choice of ϵ, the weight of at least one set drops to 0.)
7. $\quad$ Conceptually: $w^{\$} \leftarrow w^{\$} - \delta$.
8. $\quad \Delta \leftarrow \Delta + \epsilon$.
9. $\quad$ Select a set S_l such that $w(S_l) = 0$.
10. $\quad$ For all $i \neq l:\ w(S_i) \leftarrow w(S_i) + \Delta \cdot |S_i \cap S_l|$.
11. $\quad$ Add S_l to the solution and remove the elements in S_l from the problem instance. Discard sets that become empty.

In each iteration every set receives a budget proportional to its current size, and the set minimizing the ratio between its weight and size is selected. The retraction of checks ensures that at all times the ratio *weight/size* of a given set is the ratio that would be calculated in the original greedy algorithm minus Δ, and therefore the set minimizing this ratio is the same in both algorithms. Thus, Algorithm **SC** is indeed a restatement of the greedy algorithm.

As mentioned earlier, the analysis uses a new weight function $w^{\$}$. Denoting by OPT and $\text{OPT}^{\$}$ the optimum value with respect to w and $w^{\$}$, we have $\text{OPT}^{\$} = r \cdot \text{OPT}$. We first show that if $w^{\$}$ remains non-negative throughout the execution of the algorithm, the solution is r-approximate. Then we show that $r = H_n$, where $n = |U|$, ensures this. This implies an approximation guarantee of H_n.

Let a notation with subscript j denote the appropriate object in the jth iteration. Also, let $|S_i|_j$ be the size of S_i in the jth iteration. Let t be the number of iterations and let $j(u)$ denote the iteration in which element u is covered. When element u gets covered by a set S, it retracts the checks it has issued to other sets and gets deleted from the problem instance. Thus the amount u ends up actually paying is the total sum of checks it has passed to S, i.e., $\sum_{j=1}^{j(u)} \epsilon_j$. Since the solution returned by the algorithm is fully funded by these payments its cost is $\sum_{u \in U} \sum_{j=1}^{j(u)} \epsilon_j = \sum_{j=1}^{t} |U_j| \cdot \epsilon_j$. Now consider $\text{OPT}^{\$}$. In the jth iteration the cost with respect to $w^{\$}$ of every set S_i decreases by $\epsilon_j \cdot |S_i|_j$. Consequently, since every solution must cover all elements, $\text{OPT}^{\$}$ decreases by at least $\epsilon_j \cdot |U_j|$. The deletion of elements further decreases $\text{OPT}^{\$}$ since $w^{\$}$ is always nonnegative. Thus the total decrease in $\text{OPT}^{\$}$ is at least $\sum_j \epsilon_j \cdot |U_j|$, and this decrease brings $\text{OPT}^{\$}$ down from its initial value $r \cdot \text{OPT}$ to zero (the optimum value for the empty instance). Thus $\sum_{j=1}^{t} |U_j| \cdot \epsilon_j \leq r \cdot \text{OPT}$.

We have thus shown that if $w^{\$}$ remains nonnegative, the solution is r-approximate. It remains to find the smallest r ensuring this. Finding r is an optimization problem, which (following Jain et al. [15]) we tackle in the follow-

ing manner. For a given set S, the total decrease in $w^\$(S)$ equals the total sum of the checks given to S (including ones that were eventually retracted). Denote the elements of S by $u_1, \ldots, u_d$, indexed in the order by which they were covered, and let x_i be the total sum of the checks given to S by u_i. A sufficient condition on r is $r \geq \frac{1}{w(S)} \sum_{i=1}^{d} x_i$ for all possible choices of d, $w(S)$, and the x_is consistent with the algorithm, so we need to examine a worst case scenario in which the ratio is maximized. What constraints can we impose on the x_is? One set of constraints (which turns out to be all we need) is the following. Consider the moment u_i gets covered (just before any check retractions occur). Till that moment S has received an as yet unretracted sum of x_i from each of the $d-i+1$ elements $u_i, u_{i+1}, \ldots, u_d$. Since no set is ever in possession of checks exceeding its cost, we get $x_i \leq w(S)/(d-i+1)$. Thus for fixed d and $w(S)$, our optimization problem is $\max\{\frac{1}{w(S)} \sum_{i=1}^{d} x_i : \forall i, x_i \leq \frac{w(S)}{d-i+1}\}$. Clearly the optimum value is $\sum_{i=1}^{d} \frac{1}{d-i+1} = H_d \leq H_n$, which is our approximation guarantee.

Put abstractly, our combinatorial framework is the following. First establish (conceptually) a second weight function $w^\$ = r \cdot w$. Then manipulate the weights of the objects in the problem instance such that a zero cost feasible solution S emerges. At the same time manipulate (conceptually) the $w^\$$ weights accordingly so that $\textsc{Opt}^\$$ decreases by at least $w(S)$ but never becomes negative. If r is chosen such that this is indeed possible, S will be r-approximate.

Naturally, actual implementations of the framework have more structure built into them. In Algorithm **SC** we attributed the weight subtractions to the elements in such a way that every element was made responsible for a well defined contribution to the change in w and $w^\$$. The decrease in w and $w^\$$ in the ith iteration was done uniformly (from the perspective of the elements), ensuring that the decrease in $\textsc{Opt}^\$$ could be charged to the surviving elements, at least ϵ_i to each. The increases in w in the same iteration guaranteed that no element ended up contributing more than ϵ_i (in that iteration) towards covering the cost of the solution. Thus the elements forge the link between the "budget" supplied by $\textsc{Opt}^\$$ and the cost of the solution. This approach is quite straightforward and works well for covering problems. It can be expected that more sophisticated charging schemes will be developed to cope with non-covering problems.

2.2 Discussion

Consider the following LP relaxation of set cover, and its dual:

$$\begin{array}{lll} \min \sum_{i=1}^{m} w(S_i)x_i & & \max \sum_{i=1}^{m} y_u \\ \text{s.t.} \ \sum_{i:u\in S} x_i \geq 1 & \forall u \in U & \text{s.t.} \ \sum_{u\in S} y_u \leq w(S_i) \ \ \forall 1 \leq i \leq m \\ x_i \geq 0 & \forall 1 \leq i \leq m & y_u \geq 0 \qquad\qquad \forall u \in U \end{array}$$

Algorithm **SC** can be seen as a dual fitting algorithm by changing Line 2 to "Conceptually: set all dual variables to 0" and Line 7 to "Conceptually: $\forall u \in U,\ y_u \leftarrow y_u + \epsilon$." By the same arguments as in our analysis of the algorithm, it is not difficult to see that the cost of the solution equals the value of the dual objective function (y_u is the amount paid by u to the set that covers it first).

Similarly, the problem of finding the smallest r such that y/r is feasible can be formulated as the LP we have developed in the previous section [15].

The relation between the dual fitting interpretation of Jain et al. [15] and our combinatorial interpretation is no coincidence. Bar-Yehuda and Rawitz [6] showed (in the context of the primal-dual schema) that increasing a dual variable by ϵ is equivalent to subtracting the weight function obtained by multiplying the coefficients of the corresponding primal constraint by ϵ from the primal objective function. Here this is manifested as the equivalence between increasing the dual variables and subtracting δ from $w^\$$. Also, dividing y by r is equivalent (feasibility-wise) to multiplying the right hand sides of the dual constraints by r, which is equivalent to multiplying the primal objective function by r.

Thus, the translation of a dual fitting algorithm to a purely combinatorial one is as follows. First define $w^\$ = r \cdot w$. Then, every time a dual variable is increased by ϵ, multiply the coefficients of the corresponding primal constraint by ϵ and subtract them from $w^\$$. The dual fitting analysis argues that the (infeasible) dual solution upper bounds the primal one. The same proof, properly rephrased, shows that the decrease in $\textsc{Opt}^\$$ upper bounds the solution's cost. The second part of the dual fitting analysis is to find r such that dividing the dual solution by r makes it feasible. The proof of this can be used to show instead that $w^\$$ remains non-negative.

We remark that although the problem of finding the best r is formulated as an LP (actually, as a family of LPs—one for each choice of d), this is merely a convenient way to express it. We have made no use of the theory of linear programming in its solution. Thus the term *factor revealing LP* applied by Jain et al. [15] to this LP is a bit inexact, as no real linear programming is involved.

2.3 Metric Uncapacitated Facility Location

In the *uncapacitated facility location* problem we are given a set F of *facilities*, and a set C of *cities*. Each facility i has an *opening cost* $w(i) \geq 0$, and each pair of facility i and city j is associated with a *connection cost* $w(j, i) \geq 0$. Our goal is to open some facilities and connect each city to one of them while minimizing the total cost. In the *metric uncapacitated facility location* problem (MUFL) the connection costs satisfy the triangle inequality.

Jain et al. [15] restated MUFL as a special case of *set cover* in the following terms. A *star* S consists of a single facility i and a subset C' of the cities. The cost of S is $w_S = w(i) + \sum_{j \in C'} w(j, i)$. The problem is to find a minimum weight set of stars containing all cities. Jain et al. [15] describe an algorithm for MUFL that is essentially Algorithm **SC**. They show that it returns 1.861-approximate solutions on MUFL instances. They obtain this low factor by adding constraints (peculiar to MUFL) to the problem of finding the best r. The additional constraints stem from the triangle inequality and from the fact that the contribution of each city participating in a star can be partitioned into payment for its connection to the facility and payment for opening the facility, and no city ever pays for the connection of another city. (See [15].)

Jain et al. [15] also describe a second dual fitting algorithm for MUFL achieving a factor of 1.61. In this algorithm cities may "change their minds" about which facility to connect to, and when they do so they transfer funds from one star to another. This algorithm too can be analyzed in purely combinatorial terms in the manner outlined in this section. (Despite the transferring of funds between stars, the key point remains. In the ith iteration each city is responsible for a decrease of ϵ_i in $\text{OPT}^{\$}$ and is charged ϵ_i for the cost of the solution.)

We remark that Jain et al. [15] do not use linear programming in the "factor revealing" stage of either algorithm. They do, however, recount solving the LP for small values of d in order to get an estimate of the general optimal solution.

3 Primal Fitting

In [16] Jain and Vazirani present a primal-dual 3-approximation algorithm for MUFL in which they relax the primal complementary slackness conditions rather than the dual conditions. In a sense, this is the mirror image of the standard primal-dual approach, but there is another way to view it. The conceptual difference between primal-dual and dual fitting is that in standard primal-dual algorithms the "budget" supplied by the dual solution is inflated in order to cover the cost of the primal solution, whereas in dual fitting algorithms the unshrunk dual covers the weight of the primal solution (but only by shrinking it does it become feasible). In Jain and Vazirani's algorithm the dual solution covers the weight of an *infeasible* primal solution which is then inflated to make it feasible. (This, of course, is our interpretation of their algorithm. It is not how they describe it.) This idea seems befitting of the name *primal fitting*.

As with dual fitting, we show that primal fitting can be described in combinatorial terms, avoiding linear programming completely. We demonstrate this on Jain and Vazirani's algorithm and on a disk covering problem.

3.1 Metric Uncapacitated Facility Location

Algorithm **MUFL** below is our reformulation of Jain and Vazirani's algorithm (with minor modifications). Here too, it is convenient to imagine the cities paying for opening and connecting to facilities, but this time we prefer to think in terms of continuous, unit rate, flows of funds rather than discrete payments. The algorithm proceeds as follows. Initially, all cities are *active*. At a given moment every active city directs a stream of money towards each facility. The money flowing from city j towards facility i is first used to pay the connection cost $w(j,i)$, and is said to be *received by the connection*. When the connection cost is covered, the city is said to have *reached* the facility, and subsequently the money flow from city j to facility i is used to cover the opening cost of facility i, and is said to be *received by the facility*. Thus a typical active city will be simultaneously funding connection costs to several facilities and opening costs of others, and a typical facility will be receiving funds simultaneously from zero or more cities (that have reached it). As time passes, two types of events

occur (multiple events may occur simultaneously). The first is when the total sum received by some facility i reaches its opening cost $w(i)$. Facility i is then declared *open*[1], and all cities that have reached it, either at that exact moment, or earlier, are made inactive. The second event type is when an active city reaches an open facility. The city is then made inactive. When the last city becomes inactive, the algorithm retains a certain subset of the open facilities (chosen in a manner described below) and closes all others. Each city now connects to the *nearest* open facility (i.e., it uses the cheapest connection to an open facility), and conceptually retracts the payments it has made to all other facilities and connections to them. To describe the subset of facilities that remain open we need the following terminology. A city *contributes* to a facility at a given moment if the facility has received from it a nonzero amount of money. Two facilities *conflict* whenever a city contributes to both. A set of facilities is *independent* if no two of them conflict. The algorithm keeps open an arbitrary maximal (under inclusion) independent subset of the open facilities. Such a set can be found by scanning the open facilities in arbitrary order and closing every facility conflicting with a previously considered facility that has not been closed.

Algorithm **MUFL** below discretizes time at the events. It keeps track of payments by subtracting weights, and of inactive cities by deleting them. It keeps track of the cities contributing to each facility i by means of sets C_i.

Algorithm MUFL

1. $C_i \leftarrow \emptyset$ for all $i \in F$.
2. While $C \neq \emptyset$ do:
3. Let ϵ be maximal such that the weight function δ defined below satisfies $w - \delta \geq 0$.
   $\forall j \in C\ \forall i \in F,\ \delta(j,i) = \min\{\epsilon, w(j,i)\}$,
   $\forall i \in F,\ \delta(i) = \sum_{j \in C}(\epsilon - \min\{\epsilon, w(j,i)\})$.
4. $w \leftarrow w - \delta$.
5. For all $i \in F$ do:
6. $C_i \leftarrow C_i \cup \{j \in C : \epsilon - \delta(j,i) > 0\}$.
7. $C \leftarrow C \setminus \{j \in C : \exists i \in F$ such that $w(j,i) = w(i) = 0\}$.
8. Let $O = \{i \in F : w(i) = 0\}$.
9. Select a maximal set $O' \subseteq O$ such that $C_{i_1} \cap C_{i_2} = \emptyset$ for all $i_1, i_2 \in O'$, $i_1 \neq i_2$.
10. Open the facilities in O', close all others, and connect each city j to a facility $\arg\min\{w_0(j,i) : i \in O'\}$, where w_0 is the original weight function.

For the analysis, define a new weight function $w^\$ = w/r$. Let us conceptually perform the same weight subtractions on $w^\$$ as on w, and let us perform the conceptual retractions in the final stage on $w^\$$ only. Let S be the solution re-

[1] Jain and Vazirani use the term *temporarily open*.

turned by the algorithm, let OPT be the optimum value, and let a notation with subscript l denote the appropriate object in the lth iteration. In the lth iteration each surviving city sends two checks, totaling ϵ_l, in the direction of every facility i. One is received by the connection between j and i, and the other by facility i. The sums written on the checks (which may be 0) are deducted from the corresponding weights. Consequently the cost of every feasible solution, and hence OPT, drops by at least $\epsilon_l \cdot |C_l|$, since every feasible solution opens facilities and connect each city to one of them. Additionally, deleting cities can only further decrease OPT. Now consider the effect of the lth iteration on $w^{\$}(S)$. The actual drop in $w^{\$}(S)$ might be quite large, but if we factor in the fund retractions of the final stage, we can "adjust" the drop in the lth iteration and charge it to the surviving cities, at most ϵ_l per city. Thus the adjusted drop is at most $\epsilon_l \cdot |C_l|$, and therefore the total drop in OPT upper bounds the total drop (minus retractions) in $w^{\$}(S)$. If r is such that the final value of $w^{\$}(S)$ is non-negative, then $w^{\$}(S) \leq \text{OPT}$ (where the inequality refers to the initial values of $w^{\$}(S)$ and OPT) since OPT never drops below zero, as w remains non-negative. This implies an approximation factor of r.

We now argue that Setting $r = 3$ guarantees this. Upon termination, the w-cost of the facilities open in S is zero. Since they are independent, they do not suffer check retractions, so their $w^{\$}$-cost is non-positive as well. The same argument cannot be applied to the connection costs, however, since cities do not necessarily connect to facilities they have reached. Nonetheless, for every city j there exists an open facility i in S such that the connection between j and i has received checks in the amount of at least $\frac{1}{3}w(j,i) = w^{\$}(j,i)$. (See Lemma 5 in [16] for a straightforward proof of this fact, based on the triangle inequality.) Since j issues checks uniformly in all directions, all connections leading out of j that are not fully funded have received the same amount. Thus, the connection chosen for j (which is of minimum cost) has certainly received at least one third of its cost, so the $w^{\$}$-cost of this connection is non-positive upon termination.

Two remarks. Our first remark regards the asymmetry in the treatment of facilities and connections, namely, that connections might be funded to as little as a third of their actual cost, whereas facilities are always fully funded. This disparity is not fundamental—the analysis remains virtually unchanged if we declare a facility open when it has received one third of its opening cost. The reason Jain and Vazirani insist on fully funded facilities relates to their use of Algorithm **MUFL** in the context of the *k-median* problem [16], where it is important to fully fund the facilities. Our second comment concerns a somewhat displeasing aspect of the algorithm, namely, that facilities first open and then may close, and cities only get connected at the very end. This is so because Jain and Vazirani apparently modeled their algorithm on the dual ascent/reverse delete pattern common to many primal-dual algorithms, but again, it is not fundamental aspect of the algorithm. A perhaps more appealing variant of the algorithm exists, which constructs a solution in a single pass by opening facilities and connecting cities on the fly. We defer the details of this algorithm to the full version of this paper.

Our approach can be generalized as follows. Define a new weight function $w^\$ = w/r$. Now, assume that w and $w^\$$ are manipulated in such a way that (1) w remains non-negative; (2) the decrease in OPT dominates the decrease in $w^\$(S)$, where S is the solution returned by the algorithm; and (3) $w^\$(S)$ drops to zero or less. Then, the solution found is r-approximate. Just as with our combinatorial "dual fitting" framework, specific applications are more structured. For example, in Algorithm **MUFL** the cities are charged individually for the drop in OPT and the drop in $w^\$(S)$.

A Primal Fitting Interpretation. Consider the following LP relaxation of MUFL and its dual:

$$\begin{array}{lll} \min \sum_{j\in C, i\in F} w(j,i)x_{ji} + \sum_{i\in F} w(i)y_i & & \max \sum_{j\in C} \alpha_j \\ \text{s.t. } \sum_{i\in F} x_{ji} \geq 1 & \forall j \in C & \text{s.t. } \alpha_j - \beta_{ji} \leq w(j,i) \quad \forall j \in C, i \in F \\ y_i - x_{ji} \geq 0 & \forall j \in C, i \in F & \sum_{j\in C} \beta_{ji} \leq w(i) \quad \forall i \in F \\ x_{ji}, y_i \geq 0 & \forall j \in C, i \in F & \alpha_j, \beta_{ji} \geq 0 \qquad \forall j \in C, i \in F \end{array}$$

We construct a dual solution (α, β) and a *fractional*, infeasible, primal solution (x, y) as follows. We use the following terminology. A facility is *open* whenever the corresponding dual constraint is tight; a facility is *reached* by a city whenever the dual constraint corresponding to the pair is tight; a city is *active* whenever no open facility is reached by it. Starting with $(\alpha, \beta) = (0, 0)$, we increase uniformly the α_js corresponding to the active cities. (When a city becomes inactive, we stop increasing its dual variable.) As the α_js increase, each β_{ji} responds to the change in α_j so as to maintain feasibility, i.e., we set it to $\max\{0, \alpha_j - w(j,i)\}$. Additionally, the x_{ij}s and y_is respond to the changes in (α, β) so that $x_{ji} = (\alpha_j - \beta_{ji})/w(j,i)$ and $y_i = \sum_{j\in C} \beta_{ji}/w(i)$. We keep going until all cities become inactive. Then we choose a maximal independent set O' of open facilities, where a set of open facilities is *independent* if for every city j there is at most one facility i in the set such that $\beta_{ji} > 0$. We now select for each city j the nearest facility in O'(breaking ties arbitrarily) and denote it $i(j)$. We zero out all primal variables y_i corresponding to facilities not in O' and all primal variables x_{ji} such that $i \neq i(j)$.

The equivalence between this description and Algorithm **MUFL** is straightforward. At every moment α_j records the amount of money directed by city j towards each of the facilities till that moment. For each facility i, β_{ji} of this amount was received by the facility and the remainder by the connection. The primal variables record the receipt of funds as well: y_i is the fraction of facility i's opening cost received by i, and x_{ji} is the fraction of the connection cost from j to i received by the connection. Clearly, the dual solution is feasible at all times. Also, the choice of O' in the final step as an independent set of open facilities ensures that the final value of (x, y) is bounded by the final value of (α, β). Finally, Lemma 5 in [16] implies that $x_{j\,i(j)} \geq 1/3$ for all cities j. Thus multiplying the primal solution by 3 (and rounding down to 1) makes it feasible.

Note that the above description is slightly different from the one given in [16]. Specifically, instead of defining the primal solution after the algorithm terminates, we construct it together with the dual solution. Thus during the first stage

of the algorithm each primal variable measures the tightness of the corresponding dual constraint—its value is the ratio between the left hand side and right hand side of the constraint. In the standard primal-dual setting (and in the dual fitting method) we only consider for the solution elements whose corresponding dual constraints are tight. In the primal fitting setting we consider elements whose corresponding dual constraints are *weakly* tight, i.e., their tightness is at least $1/r$ (in the above measure), where r is the algorithm's performance guarantee. The equivalence between primal fitting and our combinatorial framework stems from this observation and from the equivalence between increasing dual variables and subtracting weight functions.

3.2 Disk Cover

In the design phase of cellular networks the geographical area is surveyed and m potential *base-station* locations are identified. Each base-station covers a disk shaped area (centered at the station) whose radius can be set arbitrarily large (by increasing the station's output power). The cost of operating base station b with radius r is proportional to the *energy measure*: it is $w(b) \cdot r(b)^2$, where $w(b)$ is a coefficient associated with b. Given n *client stations*, the problem is to select a minimum cost set of base-stations with corresponding radii to cover all client stations. Henceforth we refer to the locations of the client stations (which are points in the plane) as *points*. Note that the set of radii can be discretized—there are at most n "interesting" radii per base-station. Therefore, no more than nm disks need be considered. We denote the set of these disks by D. We denote the center of disk d by $c(d)$, its radius by $r(d)$, its cost (or *weight*) by $w(d)$, and the number of points it covers (or *contains*) by $|d|$.

Chuzhoy developed an unpublished 9-approximation algorithm for the problem [10]. We present her algorithm here (with permission), reformulated as Algorithm **DC** below. The idea is that in each iteration every active point issues a check in the amount of ϵ to every disk containing it (initially all points are active) and when a disk becomes fully funded the points it contains become inactive. Thus when all points become inactive, they are all covered by fully funded disks. However, as usual, a point is only willing to honor the checks it has given to one of the disks. Therefore, we construct the solution by selecting a set of disks such that at least one ninth of the cost of each disk can be ascribed to payments made by the points, such that no point contributes to more than one disk. This is done by the following procedure. Let X be the set of all fully funded disks. Iteratively delete from X the disk d^* with maximum radius and every disk intersecting it, and add to the solution the smallest disk d' (which need not be in X) centered at $c(d^*)$ that covers all of the points contained in those deleted disks. Note that $r(d') \leq 3r(d^*)$ since d^* is of maximum radius in X, and thus $w(d') \leq 9w(d^*)$. Therefore, since d^* is fully funded by the points contained in it, d' is at least one ninth funded by these same points. Since all disks intersecting d^* are deleted from X, the points will not be charged more than once. Conceptually, then, all points contained in d' retract all of the checks they have issued, except that the points contained in d^* do not retract the checks they have given d'.

Algorithm DC

1. While points remain do:
2. $\epsilon \leftarrow \min\{w(d)/|d| : d \in D\}$.
3. Define the weight function δ by: $\forall d \in D,\ \delta(d) = \epsilon \cdot |d|$.
4. $w \leftarrow w - \delta$.
5. Delete all points covered by zero-weight disks.
6. $S \leftarrow \emptyset$.
7. $X \leftarrow \{d \in D : w(d) = 0\}$.
8. While $X \neq \emptyset$ do:
9. $d^* \leftarrow \arg\max\{r(d) : d \in X\}$.
10. $Y \leftarrow \{d \in X : d \text{ and } d^* \text{ intersect}\}$.
11. $X \leftarrow X \setminus Y$.
12. Add to S the minimum radius disk centered at $c(d^*)$ that covers all points contained in disks $d \in Y$.
13. Return S.

Define $w^\$ = w/9$ and assume that the weight subtractions made in Line 4 are also made with respect to $w^\$$, and that the conceptual retractions are applied only to $w^\$$. In a given iteration OPT drops by at least ϵ times the number of surviving points, and as implied by our explanation of the algorithm, $w^\$(S)$ drops by at most the same amount (when the check retractions are factored in). Also, OPT never drops below zero, and as implied by our explanation of the algorithm, $w^\$(S)$ drops to zero or less (even when the check retractions are taken into account). Thus we obtain an approximation factor of 9.

A Primal Fitting Interpretation. We use the *set cover* LP relaxation of the problem (disks are sets; points are elements). The dual variables increase uniformly, each ceasing to increase whenever a dual constraints involving it becomes tight, until all variable cease to increase. At the same time each primal variable increases to reflect the ratio between the left hand side and right hand side of the corresponding dual constraint. The set X consists initially of all disks whose corresponding variables are 1. Our analysis above can be used to show that for disks added to S, the corresponding variables are set to at least 1/9, and that if we zero out all other primal variables, the dual objective value upper bounds the primal objective value.

Acknowledgments

We thank Reuven Bar-Yehuda and Juila Chuzhoy for helpful discussions.

References

1. V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.

2. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Shieber. A unified approach to approximating resource allocation and schedualing. *Journal of the ACM*, 48(5):1069–1090, 2001.
3. A. Bar-Noy, S. Guha, Y. Katz, J. Naor, B. Schieber, and H. Shachnai. Throughput maximization of real-time scheduling with batching. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 742–751, 2002.
4. R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. *Algorithmica*, 27(2):131–144, 2000.
5. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.
6. R. Bar-Yehuda and D. Rawitz. On the equivalence between the primal-dual schema and the local-ratio technique. In *4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 2129 of *LNCS*, pages 24–35, 2001.
7. A. Becker and D. Geiger. Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83(1):167–188, 1996.
8. D. Bertsimas and C. Teo. From valid inequalities to heuristics: A unified view of primal-dual approximation algorithms in covering problems. *Operations Research*, 46(4):503–514, 1998.
9. F. A. Chudak, M. X. Goemans, D. S. Hochbaum, and D. P. Williamson. A primal-dual interpretation of recent 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Operations Research Letters*, 22:111–118, 1998.
10. J. Chuzhoy. Private Communication.
11. V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
12. M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In Hochbaum [14], chapter 4.
13. S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.
14. D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problem.* PWS Publishing Company, 1997.
15. K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani. A greedy facility location algorithm analyzed using dual-fitting with factor-revealing LP. To appear in *Journal of the ACM*, 2003.
16. K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.
17. L. Lovász. On the ratio of optimal integral and fractional covers. *Discreate Mathematics*, 13:383–390, 1975.
18. M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 2462 of *LNCS*, pages 229–242, 2002.
19. D. Shmoys. Approximation algorithms for facility location problems. In *3rd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 1913 of *LNCS*, pages 27–33, 2000.

On the Approximability of the Minimum Fundamental Cycle Basis Problem

Giulia Galbiati[1] and Edoardo Amaldi[2]

[1] Dipartimento di Informatica e Sistemistica
Università degli Studi di Pavia, 27100 Pavia, Italy
giulia.galbiati@unipv.it
[2] Dipartimento di Elettronica e Informazione
Politecnico di Milano, Piazza L. da Vinci 32, 20133 Milano, Italy
amaldi@elet.polimi.it

Abstract. We consider the problem of finding a fundamental cycle basis of minimum total weight in the cycle space associated with an undirected biconnected graph G, where a nonnegative weight is assigned to each edge of G and the total weight of a basis is defined as the sum of the weights of all the cycles in the basis. Although several heuristics have been proposed to tackle this NP-hard problem, which has several interesting applications, nothing is known regarding its approximability. In this paper we show that this problem is MAXSNP-hard and hence does not admit a polynomial-time approximation scheme (PTAS) unless P=NP. We also derive the first upper bounds on the approximability of the problem for arbitrary and dense graphs. In particular, for complete graphs, it is approximable within $4 + \epsilon$, for any $\epsilon > 0$.

1 Introduction

Let $G = (V, E)$ be an undirected graph without loops or multiple edges and with a nonnegative weight $w(e)$ assigned to each edge $e \in E$. Associated with G there is a vector space over $GF(2)$, called the *cycle space*, consisting of the edge incidence vectors of all cycles (including the null cycle) and of all unions of edge-disjoint cycles of G. If G has n vertices, m edges and p connected components, the dimension of this space is $\nu(G) = m - n + p$. A basis of the cycle space is called a *cycle basis*. Since the cycle space of a graph is the direct sum of the cycle spaces of its biconnected components, we assume G to be biconnected, i.e., G contains at least two edge-disjoint paths between any pair of vertices.

When G is connected there are special cycle bases that can be derived from the spanning trees of G. If T is an arbitrary spanning tree of G, by adding anyone of the $m - n + 1$ edges of G which do not belong to T, the so-called *chords*, one creates a cycle and the set of these $m - n + 1$ cycles form a cycle basis, which is referred to as *fundamental cycle basis*. According to [20], a cycle basis $B = \{b_1, ..., b_{m-n+1}\}$ of G is fundamental if and only if no b_i only consists of edges belonging to other cycles of B.

In this work we investigate the following combinatorial optimization problem.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 151–164, 2004.
© Springer-Verlag Berlin Heidelberg 2004

MIN-FCB: Given a weighted graph G as above, find a fundamental cycle basis B of minimum total weight, i.e., which minimizes $W(B) = \sum_{i=1}^{m-n+1} W(b_i)$ where $W(b_i)$ is the sum of the weights of all edges in cycle b_i.

Interesting applications of MIN-FCB arise, for instance, in the testing of electrical circuits [5], the generation of minimal perfect hash functions (see [8] and the references therein), the coding of ring compounds [21], the planning of complex syntheses in organic chemistry [22] as well as in cyclic timetabling [13].

While the problem of finding a general (not necessarily fundamental) cycle basis of minimum total weight is solvable in polynomial time [11], in [7] MIN-FCB is shown to be NP-hard. Although several heuristics have been proposed for its solution (see [7, 8] and references therein), to the best of our knowledge, no performance guarantees have been established and nothing is known regarding its approximability.

In this paper we prove that MIN-FCB is MAXSNP-hard and hence does not admit a polynomial-time approximation scheme unless P=NP. Using a result in [19], relying on [2], we show that, for arbitrary graphs, the problem is approximable within a factor of $2^{O(\sqrt{\log n \log\log n})}$. For dense graphs we derive a constant (respectively an $O(\log n)$) factor if the number of edges of the complement of G is $O(1)$ (respectively $O(\log n)$). In particular for complete graphs we have a constant factor of $4 + \varepsilon$, for any $\varepsilon > 0$.

Extensively studied problems related to MIN-FCB include the Minimum Communication cost spanning Tree problem (MCT), introduced in [12], and the Minimum Routing cost spanning Tree problem (MRT), see e.g. [23]. In MCT, given a complete undirected graph G with a nonnegative weight $w(e)$ assigned to each edge $e \in E$ and a nonnegative communication requirement $r(i, j)$ for each pair of vertices i, j, one looks for a spanning tree T of G which minimizes the total communication cost, i.e., the function $com_G(T) = \sum_{i,j \in V} r(i, j) w_T(i, j)$, where $w_T(i, j)$ denotes the weight of the unique path in T joining the vertices i and j. MCT is known to be approximable within $2^{O(\sqrt{\log n \log\log n})}$ even for graphs that are not complete [19]. In [23] it is shown that MCT with all requirements equal to 1, referred to as MRT, admits instead a polynomial-time approximation scheme even for graphs that are not complete. Although MIN-FCB was shown to be NP-hard by reduction from MRT [7], the two problems differ substantially. While in MRT one minimizes the sum of the weight of the paths on the tree between all pairs of vertices, in MIN-FCB the sum is taken only over the pairs of vertices corresponding to the edges of the graph that do not belong to the tree and the weights of all these non-tree edges are also included in the objective function.

The approximability of the bottleneck version of MIN-FCB, in which one looks for a fundamental cycle basis where the weight of the maximum cycle is minimum, has been recently addressed in [10].

In Section 2 we establish the main inapproximability result, while in Section 3 we derive the above-mentioned approximability upper bounds.

2 Inapproximability Result

To establish MAXSNP-hardness of MIN-FCB we proceed by L-reduction from the following version of the well-known maximum satisfiability problem for Boolean formulas, which is MAXSNP-complete [15].

MAX-2-SAT-3-B: Given a set $U = \{u_1, ..., u_v\}$ of Boolean variables and a collection $\mathcal{C} = \{C_1, ..., C_c\}$ of disjunctive clauses with exactly 2 literals (i.e., a variable or its complement) per clause and with each literal occurring in $\mathcal{C}$ at most 3 times, find an assignment of Boolean values to the variables which satisfies as many clauses as possible.

Since L-reductions preserve approximability within constant factors, an L-reduction from a MAXSNP-complete problem to MIN-FCB implies that the latter problem is MAXSNP-hard. According to [1], MAXSNP-hard problems do not admit a PTAS, i.e., they cannot be apppproximated within every constant $\alpha > 1$, unless P=NP. The reader is referred to [3, 15] and to the proof of Theorem 2 for the definition of L-reduction and for its use in this context.

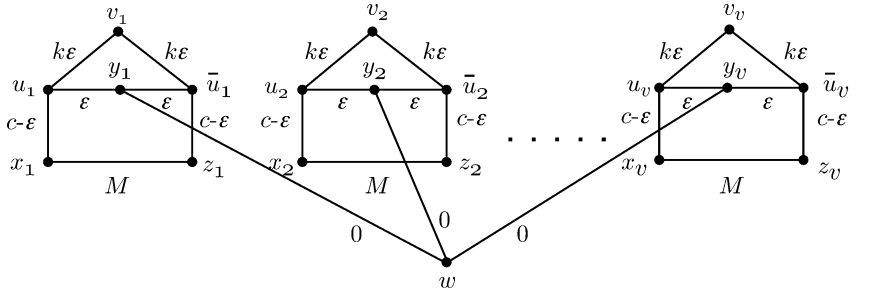


Fig. 1. Graph G'

2.1 The Reduction

For any given instance I of MAX-2-SAT-3-B, we construct a particular instance I' of MIN-FCB, i.e., we define a particular weighted biconnected graph G.

We start the construction of G from the graph $G^{'}$ given in Figure 1, where the weight assigned to each edge is indicated and the constants ε, k and M will be defined appropriately in (1) and (2). In order to obtain G from $G^{'}$, for each clause $C_i \in \mathcal{C}$ we add to $G^{'}$ two vertices c_i and c^i and the edge $\{c_i, c^i\}$ with weight equal to M. Moreover, if C_i contains the variable u_j or its complement $\bar{u}_j$, we add the edge $\{c^i, v_j\}$ with weight equal to $c - k\varepsilon$. Finally, if the variable u_j (respectively $\bar{u}_j$) occurs in clause C_i, we add the edge $\{c_i, u_j\}$ (respectively $\{c_i, \bar{u}_j\}$) with weight equal to c. The additional vertices and edges for an arbitrary clause C_i are shown in Figure 2, where only one of the dashed edges is actually included.

It is easy to verify that the resulting graph G is biconnected, with $n = 6v + 2c + 1$, $m = 8v + 5c$ so that $\nu(G) = 2v + 3c$.

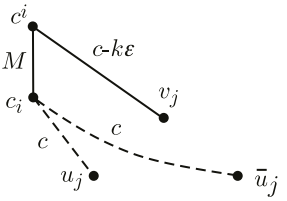


Fig. 2. Additional vertices and edges for each clause C_i

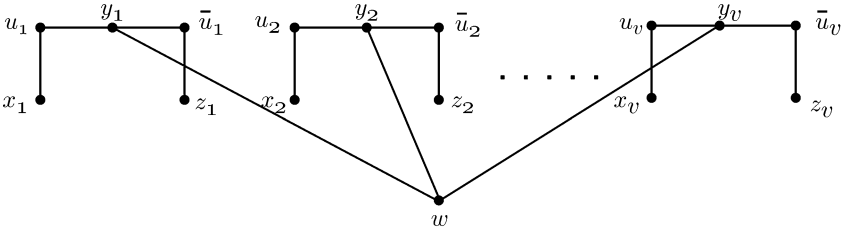


Fig. 3. Tree T'

To complete the reduction description, we need to assign values to the parameters k, ε and M. Specific values, which will yield an L-reduction (as we shall see in Subsection 2.3), are:

$$k = 4, \ \varepsilon \leq \frac{\alpha}{2}, \tag{1}$$

$$M = 2c(v + c) + 2v\varepsilon(k + 1) + 2c(2c + 2\varepsilon) + c\alpha + 1 \tag{2}$$

with α being a sufficiently small fixed constant as, for instance, $\alpha \leq 2/7$.

2.2 Well-Behaved and Quasi-Well-Behaved Trees

We now investigate the structure of some spanning trees of the graphs G constructed from the instances of MAX-2-SAT-3-B via the reduction. The purpose of this investigation is to prove Theorem 1 which plays a key role in the next subsection, where we show that the reduction is an L-reduction.

Definition 1. *A spanning tree T of G is well behaved if it satisfies the following properties:*

- *it contains all the edges of tree T' in Figure 3;*
- *for each variable u_j, $j = 1, ..., v$, it includes exactly one of the two edges $\{v_j, u_j\}$ or $\{v_j, \bar{u}_j\}$;*
- *for each clause C_i, $i = 1, ..., c$, it includes $\{c^i, v_j\}$, for some $j = 1, ..., v$, and either $\{c_i, u_j\}$ or $\{c_i, \bar{u}_j\}$, depending on whether C_i contains the literal u_j or $\bar{u}_j$.*

First we notice that by definition a well-behaved tree T does not include edges of weight M. Moreover, for each clause C_i the corresponding edge $\{c_i, c^i\}$ belongs to a cycle in G of weight $M+2c$ or $M+2c+2\varepsilon$, so that the number c of clauses can be partitioned into two sets of cardinality c_y and c_n, depending on whether $\{c_i, c^i\}$ belongs to a cycle of the first or second type. Clearly $c = c_y + c_n$ and it is not difficult but somewhat lengthy to verify that the total weight of the fundamental basis associated to a well-behaved tree T, denoted by $fund_G(T)$, satisfies

$$fund_G(T) = F + 2\varepsilon c_n \tag{3}$$

with $F = (M+2c)(v+c) + 2v\varepsilon(k+1) + 2c(2c+2\varepsilon)$.

Then we observe that from any well-behaved tree T of G it is possible to derive a truth assignment for I that satisfies either the c_y or the c_n clauses. Indeed, we can satisfy the first (respectively second) type of clauses by setting to 1 (respectively 0) all variables u_j for which the tree T includes the edge $\{v_j, u_j\}$ and all negated variables $\bar{u}_j$ for which the tree T includes the edge $\{v_j, \bar{u}_j\}$.

Finally, note that if an instance I of MAX-2-SAT-3-B is satisfiable then it is easy to construct a well-behaved tree T having $fund_G(T)$ exactly equal to F. We will prove that this tree has a set of fundamental cycles that forms a minimum fundamental cycle basis. If I is not satisfiable, then we will show that any minimum fundamental cycle basis for the cycle space of G has a total weight differing from F by a value which allows us to measure the minimum number of unsatisfiable clauses in I.

We can now state the key result.

Theorem 1. *Every spanning tree T of G satisfying the inequality*

$$fund_G(T) \leq F + c\alpha, \tag{4}$$

with F, c and α as above, either is well-behaved or it can be transformed into a well-behaved tree T' with $fund_G(T') \leq fund_G(T)$.

A spanning tree T of G satisfying (4) can thus be viewed as a *quasi-well-behaved* spanning tree. From now on let $B = \{b_1, ..., b_{m-n+1}\}$ be the set of the fundamental cycles of such a quasi-well-behaved tree T of G.

The proof of Theorem 1 relies on the following series of properties.

Proposition 1. *The following holds:*

i) Each edge of G with weight equal to M belongs to only one fundamental cycle in B.
ii) Each fundamental cycle of T having at least one edge of weight M has one of these edges as a chord of T.
iii) There are in B at most v fundamental cycles having weight less than 1; the weight of any other cycle in B is at least $2c+2\varepsilon$.
iv) If a cycle b in B includes l edges of weight M then $W(b) \geq l(M+2c-k\varepsilon)$.

Proof. (i) holds because the value assigned to M in (2) makes $M(c+v+1) > F + c\alpha$, and (ii) follows from (i). Point (iii) simply follows by looking at G. To prove (iv) it is enough to observe that in order to include in a cycle an edge $\{c^i, c_i\}$, two edges respectively of weight $c - k\varepsilon$ and c must be used; to include instead an edge $\{x_j, z_j\}$ two edges of weight $c - \varepsilon$ must be used and $k \geq 2$.

Lemma 1. *The following holds:*

i) the number of cycles in B that include an edge of weight M is exactly $c+v$;
ii) the number of cycles in B of weight less than 1 is exactly v.

Proof. We prove (i). Because of Proposition 1 (i) this number cannot be larger than $c+v$. Suppose it is less than $c+v$, and equal to $c+v-\Delta$ for some integer $\Delta \geq 1$. We derive a lower bound on $W(B)$ larger than $F+c\alpha$, thus contradicting (4). In fact the sum of the weights of the cycles containing an edge of weight M is at least $(c+v)(M+2c-k\varepsilon)$ because of Proposition 1 (iv); moreover there are at most v cycles of weight less than 1 and the remaining n_c cycles have weight at least $2c+2\varepsilon$ because of Proposition 1 (iii). Hence we have: $W(B) \geq (c+v)(M+2c-k\varepsilon)+2v\varepsilon(k+1)+(2c+2\varepsilon)n_c$ with $n_c = \nu(G)-v-v-c+\Delta = 2c+\Delta$. The lower bound for $W(B)$ is equal to $F-k\varepsilon(c+v)+\Delta(2c+2\varepsilon)$ which is not less than $F-k\varepsilon(c+v)+(2c+2\varepsilon)$. It is easy to verify that $F-k\varepsilon(c+v)+(2c+2\varepsilon) > F+c\alpha$ if (1) holds, being $v \leq 2c$. The proof of (ii) follows similarly.

Lemma 2. *Every cycle b in B contains at most two edges having weight in $[1, c]$.*

Proof. Suppose that b has more than two of these edges. We distinguish the cases where b contains no edges of weight M or it has exactly one of these edges. In the first case we have $W(b) \geq 3c-1$ and therefore, similarly as in the previous lemma, we can write:
$W(B) \geq (3c-1)+(c+v)(M+2c)+2v\varepsilon(k+1)+(2c+2\varepsilon)(2c-1) = F+c-1-2\varepsilon$.
In the second case we have $W(b) \geq 3c-1+M$ and we can write:
$W(B) \geq 3c-1+M+(c+v-1)(M+2c)+2v\varepsilon(k+1)+(2c+2\varepsilon)2c = F+c-1$.
In both cases it is easy to see that, since (1) makes true the inequality $F+c-1-2\varepsilon > F+c\alpha$, we get the contradiction $W(B) > F+c\alpha$. Indeed, if $\varepsilon \leq \frac{\alpha}{2}$ then $1-\frac{1+\alpha}{c} \leq 1-\frac{1+2\varepsilon}{c}$; since $c \geq 3$, any α such that $\alpha < 1-\frac{1+\alpha}{3}$, like the chosen $\alpha \leq \frac{2}{7}$, satisfies the above inequality.

Lemma 3. *Every cycle b in B that contains an edge $\{c_i, c^i\}$, for some $i = 1, ..., c$, and an edge $\{c^i, v_j\}$, for some $j = 1, ..., v$, either it contains $\{c_i, u_j\}$ or $\{c_i, \bar{u}_j\}$, depending on whether u_j or $\bar{u}_j$ occurs in clause C_i, or it can be replaced in B by another cycle having this property without increasing $W(B)$.*

Proof. Suppose that u_j (respectively $\bar{u}_j$) is in clause C_i but edge $\{c_i, u_j\}$ (respectively $\{c_i, \bar{u}_j\}$)is not in cycle b; call this edge a. Because of Lemma 2 the two cases we need to consider are illustrated in Figure 4 where all edges in cycle b are drawn in heavy lines but the one belonging to the chords of T which is drawn with a light line, as the other edges of G that are known not to belong to T.

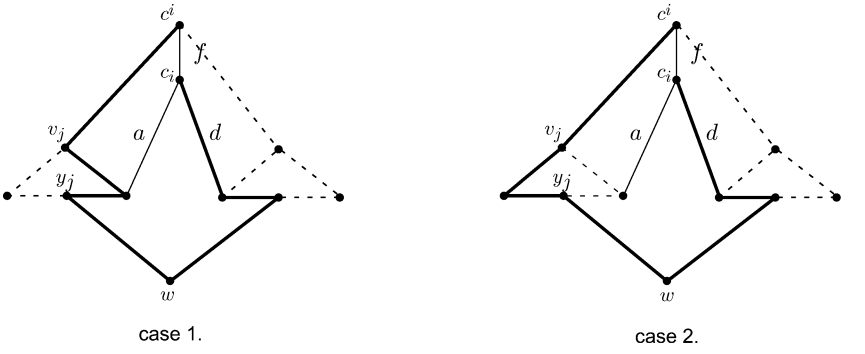


Fig. 4. The cases considered in the proof of Lemma 6

Case 1. Using Lemma 2 and the fact that every clause in instance I has two literals we may conclude that d belongs exactly to two cycles in B, having as chords respectively f and a. It is easy to verify that the insertion into T of edge a and the removal of edge d decreases $W(B)$ by 2ε.

Case 2. Two subcases need to be distinguished, which are are illustrated in Figure 5.

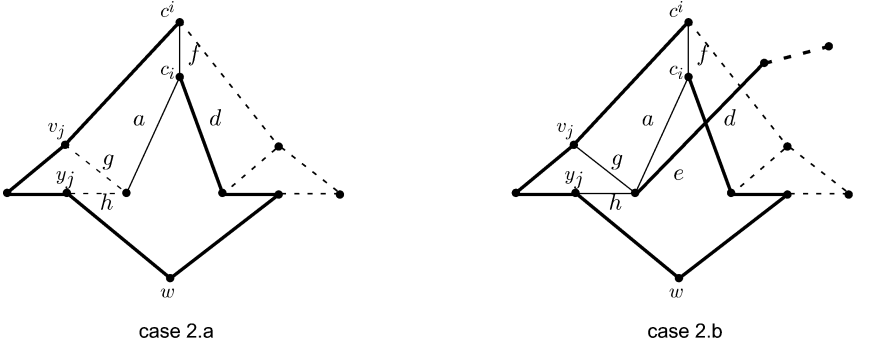


Fig. 5. The case 2 subcases

Subcase 2.a. One of the edges g and h is in T. It is easy to verify that d belongs exactly to two cycles in B and that the insertion into T of edge a and the removal of edge d does not increase $W(B)$.

Subcase 2.b. None of the edges g and h is in T. Consider the path in T from node ω to the endvertex u_j or $\bar{u}_j$ of edge a and let e be the last edge in the path. Since chord a would identify a cycle in B contradicting Lemma 2 we conclude that this case cannot happen.

Corollary 1. *Every cycle in B that contains an edge $\{c_i, c^i\}$, for some $i = 1, ..., c$, either has weight equal to $M + 2c$ or equal to $M + 2c + 2\varepsilon$.*

Lemma 4. *Every cycle b in B that contains an edge $\{x_j, z_j\}$, for some $j = 1, ..., v$, either it contains the four edges $\{x_j, u_j\}$, $\{u_j, y_j\}$, $\{y_j, \bar{u}_j\}$, $\{\bar{u}_j, z_j\}$ or it can be replaced in B by another cycle having this property without increasing $W(B)$.*

Proof. Because of Lemma 2 the cases to be considered are illustrated in Figure 6, where cycle b contains the edges $\{v_j, u_j\}$,$\{v_j, \bar{u}_j\}$ and either (Case 1) T includes one of the edges $\{u_j, y_j\}$, $\{y_j, \bar{u}_j\}$ (say $\{u_j, y_j\}$) or (Case 2) none of them.

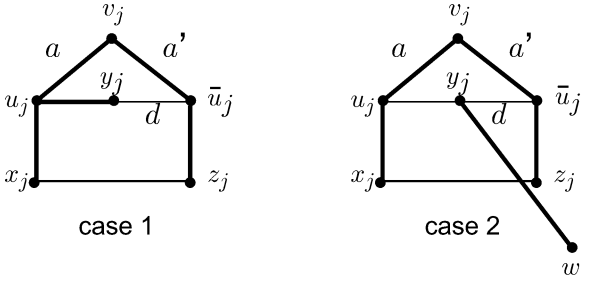


Fig. 6. The cases considered in the proof of Lemma 8

Let us first examine Case 2. Here tree T must include $\{\omega, y_j\}$. Consider the path in T from ω to $\bar{u}_j$. It must go through c^i or c_i, for some i; hence tree T must include two edges incident to c^i or c_i. Since $\{c_i, c^i\}$ is not in T, another edge of the four edges incident to c^i and c_i and of weight less than M must be in T, leaving the forth edge able to identify a cycle contraddicting Lemma 2. Hence this case cannot happen.

Now we examine Case 1. The cycles containing edge a, because of Lemma 3, only include:

- the cycle with chord $\{x_j, z_j\}$, of weight $M + 2c + 2k\varepsilon - 2\varepsilon$;
- the cycle with chord d, of weight $2\varepsilon + 2k\varepsilon$;
- cycles with chords $\{c^i, c_i\}$, for some i, of weight $M + 2c$;
- cycles with chords $\{c^i, c_i\}$, for some i, of weight $M + 2c + 2\varepsilon$;
- cycles that use $\{y_j, \omega\}$ and go through some c^i, of weight $2c+2\varepsilon$ (see Figure 7.(a));
- cycles that use $\{y_j, \omega\}$ and go through some c_i, of weight $2c + 2\varepsilon + 2k\varepsilon$ (see Figure 7.(b));

If in T we add edge d and remove edge a then the weights of the cycles in the list above become, respectively, $M+2c$, $2\varepsilon+2k\varepsilon$, $M+2c+2\varepsilon$, $M+2c+2\varepsilon$, $2c+2\varepsilon$, $2c + 2\varepsilon$. Hence only the weights of the cycles in the third group increase but the number of these cycles is bounded above by 3. If we require that the maximum

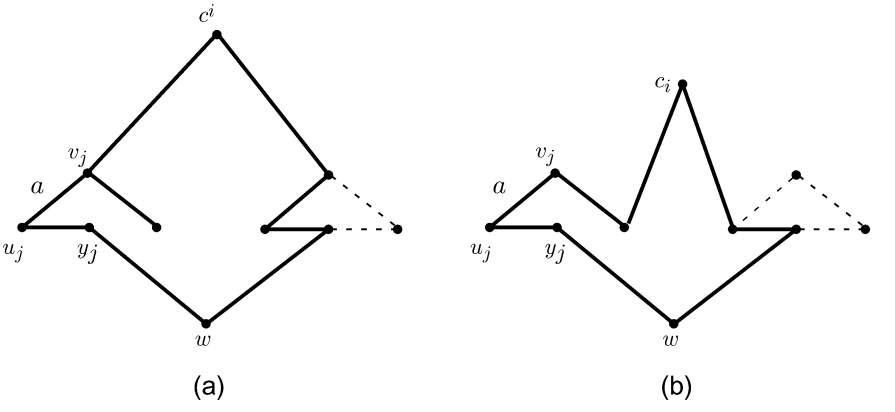


Fig. 7. The case 1 subcases

increase is not larger than a minimum decrease, i.e., that $2\varepsilon\ 3 \leq 2k\varepsilon - 2\varepsilon$ and hence $k \geq 3 + 1$ like in (1), the conclusion follows.

Proof of Theorem 1. We first transform T to satisfy the properties required by Lemma 3 and Lemma 4. Then we observe that all edges in tree $T^{'}$ of Figure 3, but the $\{\omega, y_j\}$ for all $j = 1, ..., v$, must be in T, because of Proposition 1 and Lemma 4. Using Corollary 1, Lemma 3 and a reasoning similar to the one used to prove Case 2 in Lemma 4, we conclude that edges $\{\omega, y_j\}$, for all j, are in T. Hence tree $T^{'}$ of Figure 3 is in T. Finally, Lemma 3 and Corollary 1 imply that tree T also satisfies the second and the third properties of a well-behaved tree.

2.3 MAXSNP-Hardness

We now prove that the polynomial-time reduction given in Subsection 2.1 yields an L-reduction and hence that MIN-FCB is MAXSNP-hard. We start with the following result.

Lemma 5. *An instance I of* MAX-2-SAT-3-B *has at most η unsatisfiable clauses iff there exists in the associated graph G a spanning tree T with $fund_G(T) \leq F + 2\varepsilon\eta$, where $F = (M + 2c)(v + c) + 2v\varepsilon(k + 1) + 2c(2c + 2\varepsilon)$.*

Proof. If an instance I has η unsatisfiable clauses it is easy to find the required spanning tree. Conversely, suppose that a spanning tree T has $fund_G(T) \leq F + 2\varepsilon\eta$. If $\eta \geq c$ then the conclusion is obvious since every instance has at most c unsatisfied clauses. If instead $\eta < c$, since $\varepsilon \leq \alpha/2$, we have $fund_G(T) < F + c\alpha$ and Theorem 1 applies. For the well-behaved tree $T^{'}$ that exists according to the theorem, we know from (3) that $fund_G(T^{'}) = F + 2\varepsilon c_n \leq F + 2\varepsilon\eta$ so that the number of unsatisfied clauses in I is at most η.

Thus there exists a simple relation between the maximum number of satisfiable clauses in instance I, denoted by $opt(I)$, and the minimum total weight of a fundamental cycle basis of instance I', denoted by $opt(I')$. In particular, if $opt(I) = c_y^*$ then $opt(I') = F + 2\varepsilon(c - c_y^*)$.

Theorem 2. MIN-FCB *is* MAXSNP*-hard and therefore it does not admit a polynomial-time approximation scheme, unless* P=NP.

Proof. It suffices to exhibit an L-reduction $(t_1, t_2, \beta_1, \beta_2)$ from MAX-2-SAT-3-B to MIN-FCB, where, as defined in [3], t_1 and t_2 are two appropriate polynomially computable functions and β_1 and β_2 are two positive constants. As t_1 we use the reduction described in Subsection 2.2, which associates to any instance I of MAX-2-SAT-3-B a particular instance I' of MIN-FCB. The function t_2, which associates to any instance I of MAX-2-SAT-3-B and any spanning tree T of the corresponding instance I' of MIN-FCB a feasible solution $t_2(I,T)$ of I, is defined depending on the value of $fund_G(T)$. If $fund_G(T) \leq F + c\alpha$ then $t_2(I,T)$ is the truth assignment, which is derivable from the well-behaved tree T' that exists according to Theorem 1, satisfying the c_y clauses of I as explained in Subsection 2.2. Otherwise $t_2(I,T)$ is any truth assignment for I that satisfies more than $c/2$ clauses.

The two following inequalities need to be verified:

1. $opt(I') \leq \beta_1\ opt(I)$
2. $|opt(I) - val(I, t_2(I,T))| \leq \beta_2\ |fund_G(T) - opt(I')|$,

where for any given truth assignment τ for instance I, $val(I,\tau)$ denotes the number of satisfied clauses.

The first inequality follows directly from the fact that if $opt(I) = c_y^*$ then $opt(I') = F + 2\varepsilon(c - c_y^*)$. Indeed k, ε , α are positive fixed constants and $v \leq 2c$ because each clause contains two variables and $c \leq 2opt(I)$ since at least half of the clauses of any instance can always be satisfied.

To verify the second inequality we proceed as follows. If $fund_G(T) \leq F + c\alpha$ and T' is defined as above we have that $fund(T') = F + 2\varepsilon c_n$ and $c_y = c - c_n$. Hence we have that $|fund_G(T) - opt(I')| \geq |fund_G(T') - opt(I')| = 2\varepsilon|c_y - c_y^*| = 2\varepsilon|opt(I) - val(I, t_2(I,T))|$ and 2. holds with $\beta_2 = \frac{1}{2\varepsilon}$.

If $fund_G(T) > F + c\alpha$ the definition of function t_2 implies in this case $|opt(I) - val(I, t_2(I,T))| \leq \frac{c}{2}$. We derive easily that $\frac{c}{2} = \frac{1}{2\varepsilon}(\varepsilon c) \leq \frac{1}{2\varepsilon}(2\varepsilon c_y^*) \leq \frac{1}{2\varepsilon}(2\varepsilon c_y^* + c\alpha - 2\varepsilon c) = \frac{1}{2\varepsilon}(c\alpha - 2\varepsilon c_n^*)$. From the lower bound on $fund_G(T)$ it follows that $|fund_G(T) - opt(I')| > c\alpha - 2\varepsilon c_n^*$ which concludes the proof.

3 Approximability Results

We now turn to some upper bounds on the approximability of MIN-FCB that derive from recent results regarding related problems. We first consider the general case of arbitrary weighted graphs.

Proposition 2. MIN-FCB *is approximable within* $2^{O(\sqrt{\log n \log\log n})}$.

Proof. When the problem is restricted to instances with all weights equal to 1, the above approximability bound is a straighforward consequence of the main result in [2]. In the general case of arbitrary weighted graphs, we proceed by reduction from MIN-FCB to MCT and we use the polynomial-time approximation algorithm for MCT given in [19]. Let $G = (V, E)$ be the graph of an arbitrary instance of MIN-FCB. For each pair (i, j) of vertices of G, a requirement $r(i, j)$ is specified as follows: $r(i, j) = 1$ if i and j are the end vertices of an edge of E and $r(i, j) = 0$ otherwise. Let $G^{'}$ be the graph with the added requirements, which is a particular instance of MCT on graphs that are not necessarily complete. For any subset $S \subseteq E$, let $W(S) = \sum_{e \in E} w(e)$.

For any spanning tree T (of G and also of $G^{'}$) the functions $fund_G(T)$ and $com_{G'}(T)$ are well defined and it is easy to verify that they satisfy:

$$com_{G'}(T) = fund_G(T) - W(E) + 2W(T). \tag{5}$$

Now, let T_c^* (respectively by T_f^*) be a spanning tree that minimizes the function $com_{G'}(T)$ (respectively $fund_G(T)$). Since $com_{G'}(T_c^*) \leq com_{G'}(T_f^*)$ and for any subset $S \subseteq E$ we have $W(S) \leq fund_G(T_f^*)$ relation (5) directly implies the following.

Claim 1 $fund_G(T_c^*) \leq 3 fund_G(T_f^*)$.

Claim 2 *If* MCT *with requirements in* $\{0, 1\}$ *is approximable within* $\rho > 1$, *then* MIN-FCB *is approximable within* $3\rho + 1$.

Let T_c' be a spanning tree of $G^{'}$ that solves MCT within ρ, i.e., $com_{G'}(T_c') \leq \rho\ com_{G'}(T_c^*)$. We derive that:

$$\begin{aligned} fund_G(T_c') &= com_{G'}(T_c') + W(E) - 2W(T_c') \leq \rho\ com_{G'}(T_c^*) + W(E) - 2W(T_c') \\ &\leq \rho\ com_{G'}(T_f^*) + W(E) - 2W(T_c') \\ &\leq \rho\ (fund_G(T_f^*) - W(E) + 2W(T_f^*)) + W(E) - 2W(T_c') \\ &\leq \rho\ fund_G(T_f^*) + 2\rho W(T_f^*) + W(E) \leq (3\rho + 1) fund_G(T_f^*) \end{aligned}$$

and this shows that T_c' is a $3\rho + 1$-approximate solution of MIN-FCB.

Since in [19] it is proved that MCT with arbitrary graphs and arbitrary requirements is approximable within $2^{O(\sqrt{\log n \log\log n})}$, the conclusion follows from Claim 2.

Better upper bounds on the approximability of MIN-FCB can be derived for dense graphs.

Proposition 3. MIN-FCB *is approximable within* $O(1)$ *if* $m_L = O(1)$ *and within* $O(\ln n)$ *if* $m_L = O(\lg n)$, *where* m_L *denotes the number of edges in the complement of the graph* G.

Proof. The proof is similar to that of Proposition 2. Instead of the polynomial-time approximation algorithm for MCT with arbitrary graphs and requirements,

we use here the polynomial-time approximation scheme for MCT with arbitrary graphs but all requirements equal to 1 (MRT), presented in [23]. Therefore, given the graph G of an arbitrary instance of MIN-FCB, we set all requirements $r(i,j)$ in G' equal to 1. The analogue of (5) now becomes:

$$com_{G'}(T) = fund_G(T) - W(E) + 2W(T) + \sum_{\{i,j\} \notin E} w_T(i,j), \tag{6}$$

where $w_T(i,j)$ denotes the weight of the unique path in tree T joining vertices i and j. According to the analogue of Claim 1 we have $fund_G(T_c^*) \leq (3 + m_L) fund_G(T_f^*)$. According to the analogue of Claim 2, if MCT with uniform requirements is approximable within ρ then MIN-FCB is approximable within $\rho(3+m_L)+1$. Since in [23] it is proved that MRT is approximable within $(1+\varepsilon)$, for every $\varepsilon \geq 0$, then MIN-FCB is approximable within $(1+\varepsilon)(3+m_L)+1$, and the conclusions follow both for $m_L = O(1)$ and $m_L = O(\lg n)$.

Proposition 3 implies therefore that MIN-FCB, restricted to complete graphs, is approximable within a constant factor of $4+\varepsilon$, for any $\varepsilon > 0$, since $m_L = 0$. This is in particular true for metric graphs, which are complete and whose edge weights satisfy the triangle inequality.

The question naturally arises as to whether the recent results on the approximability of finite metrics by tree metrics (see e.g. [4, 6, 14, 9]) imply stronger approximability upper bounds for MIN-FCB. In particular in [9] it is shown that given any metric graph $M = (V, E)$ with additional "weights" $r(i,j)$ on the edges of M, one can find in polynomial-time a single tree T such that for all i, j in V, we have $d_M(i,j) \leq d_T(i,j)$ and

$$\sum_{i,j \in V} r(i,j)\, d_T(i,j) \leq O(\log n) \sum_{i,j \in V} r(i,j)\, d_M(i,j) \tag{7}$$

where $d_M(i,j)$ denotes the length of a shortest path between nodes i and j in graph M and $d_T(i,j)$ the length of the unique path between i and j in tree T. Although the above problem is closely related to MIN-FCB, this result does not imply a better approximation factor for MIN-FCB since the tree found is not guaranteed to be a spanning tree of the original graph G. This property is clearly true for metric graphs which are complete. But in this case the above-mentioned result implies a logarithmic approximation factor which is weaker than the constant factor of Proposition 3.

4 Concluding Remarks

We have presented the first results regarding the approximability of the problem of finding fundamental cycle bases of minimum total weight in undirected biconnected graphs. On the one hand, we have proved that MIN-FCB is MAXSNP-hard and hence does not admit a polynomial-time approximation scheme, unless

P=NP. Thus MIN-FCB turns out to be harder to approximate than the related problem MRT [23]. On the other hand, we have derived the first upper bounds on its actual approximability factor for arbirary graphs as well as for dense graphs. We leave as an open question whether MIN-FCB is approximable within a logarithmic factor for arbitrary graphs.

Acknowledgments

The authors would like to thank Francesco Maffioli for many helpful discussions.

References

1. Arora S., Lund C., Motwani R., Sudan M., Szegedy M.: Proof verification and hardness of approximation problems. Journal of ACM **45** (1998) 501–555.
2. N. Alon, Karp R., Peleg D., West D.: Graph-theoretic game and its application to the k-server problem. SIAM J. Comput. **24** (1995) 78–100.
3. Ausiello G., Crescenzi P., Gambosi G., Kann V., Marchetti-Spaccamela A., Protasi M.: Complexity and approximation: Combinatorial optimization problems and their approximability properties. Springer-Verlag 1999.
4. Bartal Y.: On the approximation of metric spaces by tree metrics. Proceedings of STOC 1998, 161–168.
5. Brambilla A., Premoli A.: Rigorous event-driven (RED) analysis of large scale RC circuits. IEEE Trans. on CAS-I **48** (2001) 938–954.
6. Charikar M., Chekuri C., Goel A., Guha S., Plotkin S.: Approximating a finite metric by a small number of tree metrics. Proceedings of FOCS 1998, 161–168.
7. Deo N., Prabhu G.M., Krishamoorthy M.S.: Algorithms for generating fundamental cycles in a graph. ACM Trans. Math. Software **8** (1982) 26–42.
8. Deo N., Kumar N., Parsons J.: Minimum-length fundamental-cycle set: New heuristics and an empirical study. Congressus Numerantium **107** (1995) 141–154.
9. Fakcharoenphol J., Rao S., Talwar K.: A tight bound on approximatinng arbitrary metrics by tree metrics. Proceedings of STOC 2003, 448–455.
10. Galbiati G.: On min-max cycle bases. Proceedings of ISAAC 2001, P. Eades and T. Takaoka (Eds.), LNCS 2223, Springer-Verlag (2001) 116–123.
11. Horton J.D.: A polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM J. Comput. **16** (1987) 358–366.
12. Hu T.C.: Optimum communication spanning trees. SIAM J. Comput. **3** (1974) 188–195.
13. Liebchen C. and Peeters L.: On cyclic timetabling and cycles in graphs. Technical Report 761, Technische Universität Berlin, 2002.
14. Konjevod G., Ravi R., Salman F. S.: On approximating planar metrics by tree metrics. Information Processing Letters **80** (2001) 213–219.
15. Papadimitriou C. H., Yannakakis, M.: Optimization, approximation, and complexity classes. J. Comput. System Sci. **43** (1991) 425-440.
16. Peleg D.: Polylogarithmic approximation for minimum communication spanning trees. Technical Report CS 97-10, The Weizmann Institute, July 1997.
17. Peleg D.: Low stretch spanning trees. Proc. 27th Int. Symposium on Math. Foundations of Computer Science, LNCS **1443** (2002) 68–80.

18. Peleg D., Reshef E.: Deterministic polylog approximation for minimum communication spanning trees. LNCS **1443** (1998) 670–681.
19. Reshef E.: Approximating minimum communication cost spanning trees and related problems. M.Sc. Thesis, The Weizmann Institute of Science, 1999.
20. Syslo M.M.: On cycle bases of a graph. Networks **9** (1979) 123–132.
21. Sussenouth E. Jr: A graph theoretical algorithm for matching chemical structures. J. Chem. Doc. **5** (1965) 36–43.
22. P. Vismara: Reconnaissance et représentation d'éléments structuraux pour la description d'objets complexes. Application à l'élaboration de stratégies de synthèse en chimie organique. Thèse de Doctorat, Université de Montpellier II, France, 1995.
23. Wu B. Y., Lancia G., Bafna V., Chao K.-M., Ravi R., Tang, C. Y.: A polynomial-time approximation scheme for minimum routing cost spanning trees. SIAM J. Comput. **29** (1999) 761–778.

The Pledge Algorithm Reconsidered under Errors in Sensors and Motion

Tom Kamphans and Elmar Langetepe

University of Bonn
Institute of Computer Science I
Römerstraße 164
53117 Bonn
Germany
{kamphans,langetep}@cs.uni-bonn.de
http://web.cs.uni-bonn.de/I/agklein.html

Abstract. We consider the problem of escaping from an unknown polygonal maze under limited resources and under errors in sensors and motion. It is well-known that the pledge algorithm always finds a path out of an unknown maze without any means of orientation, provided that such a path exists. The pledge algorithm sums up the turning angles along the boundary of the obstacles and plans its way by using a single counter and no further information. The correctness proof makes use of the fact that motions in the free space and the measurement of turning angles can be done exactly. We consider the case that the free motion of the robot and the computation of turning angles are erroneous.
This work is a first attempt to incorporate error assumptions into theoretically sound proofs of online motion planning algorithms.

1 Introduction

Online motion planning in unknown environments is theoretically well-understood and practically solved in many settings. During the last decade many different objectives where discussed under several robot models.

For instance, the task of searching for a target using a touch sensor and a compass to the goal was first considered by Lumelsky and Stepanov [12]. They invented the BUG algorithm and many variants of this algorithm were discussed and analyzed afterwards, for example see Rajko and La Valle [14] and Kamon and Rivlin [10]. A BUG-like algorithm was also used for the Mars-Rover project, see Laubach [11]. The correctness of the algorithms are proven under idealistic assumptions whereas the practical relevance is always tested empirically.

Moreover, the task of exploring an unknown environment has attracted theoretical and practical attention. Deng et. al. [4] studied the case of a robot equipped with a vision system that maps an unknown, simple, rectangular environment. They developed a strategy and prove that under correct vision and motion the corresponding path of the robot is never longer than $\sqrt{2}$ times the optimal path computed under full information in advance. The strategy is called

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 165–178, 2004.
© Springer-Verlag Berlin Heidelberg 2004

$\sqrt{2}$-*competitive* in this model. Under the same idealistic assumptions Hoffmann et. al. [8] presented and analyzed an 26.5-competitive algorithm for exploring the interior of a simple polygon. On the other hand the task of building a map of an unknown environment is solved practically and analyzed empirically in the field of robotics, see e. g. Batalin and Sukhatme [2] or Yamauchi et. al. [17]. For a general overview of theoretical online motion planning problems see the surveys [15, 16, 3, 13, 9].

Theoretical correctness results and performance guarantees from labyrinth theory or computational geometry suffer from idealistic assumptions, therefore in the worst case a correct implementation is impossible. On the other hand practioners analyze correctness results and performance guarantees mainly statistically. So it might be useful to investigate, how online algorithms with idealistic assumptions behave, if those assumptions cannot be fulfilled, more precisely, can we incorporate assumptions of errors in sensors and motion directly into the theoretical analysis?

As a first approach we consider a simple and theoretically sound online strategy, the well-known pledge algorithm, see Abelson and diSessa [1] and Hemmerling [7]. Given an unknown polygonal maze, the robot has to leave the maze, using nothing else than a touch sensor, the ability to measure its turning angles and a very limited amount of memory. The robot is not able to create a map or to set landmarks. The pledge algorithm assumes that the robot is able to move a straight line between the obstacles and to count its turning angles correctly. Gritzmann [5] remarked that it would be interesting to know how the pledge algorithm behaves, if those assumptions cannot be fulfilled. Of course, if the robot can make arbitrary big mistakes, there are always environments, in which the robot is hopelessly trapped. But what if the robot's errors are small enough? Are there upper bounds for measuring errors? We investigate which conditions must hold to ensure that a robot can leave an unknown maze with a pledge-like algorithm.

The paper is organized as follows. In Sect. 2 we specify the robot's task and the model of the robot and its environment. We recapitulate the pledge algorithm, and analyze the sources for errors in this algorithm. With this insight, in Sect. 3 we define a class of curves in the robot's environment that fulfill a set of conditions. Further we show that a robot will escape from an unknown maze, if its path somehow *follows* a curve from that class. Last, in Sect. 5 we discuss the impact of our results on the design of a robot. Our main result is, that a robot equipped with a compass with reasonable small errors is able to leave an unknown maze using the pledge algorithm; see Sect. 4.1. Further we show results for a robot with exact free motion and bounded angle measuring error and for a robot in an 'almost rectangular' environment.

2 Preliminaries

Let us assume that a maze with polygonal shaped obstacles in the plane is given. The robot is able to recognize and follow a wall in a specified direction (w. l. o. g.

counter-clockwise) and to count the turning angles. How these abilities can be realized depends on the hardware of a specific robot. For example, the turning angles may be counted by odometry or by measuring angles along the walls with sensors. Other abilities for orientation and navigation are not required, especially it is not necessary that the robot can build a map of its environment.

The task of leaving an unknown maze can be solved using the well-known pledge algorithm, see algorithm 1, which performs only two types of movements: Either the robot follows the wall of an obstacle and counts the turning angles, or the robot moves through the free space between the obstacles in a fixed direction. The latter task always starts at vertices of the obstacles when the angle-counter reaches a pre-defined value. As soon as the robot leaves the maze it recieves a signal of success.

Algorithm 1: Pledge

REPEAT
  $\omega = 0$
  REPEAT
    Move in direction ω in the free space
  UNTIL Robot hits an obstacle
  REPEAT
    Follow the wall in counter-clockwise direction
    Count the overall turning angle in ω
  UNTIL Angle Counter $\omega = 0$
UNTIL Robot is outside the maze

Both types of movements in the pledge algorithm may be afflicted with errors. Either the turning angles are not measured exactly and the robot leaves the obstacle earlier or later than expected, or the robot cannot follow its initial direction during the movement in the free space. In the idealised setting the robot is error-free and it was shown by Abelson and diSessa [1] and Hemmerling [7] that in this case a robot will escape from a polygonal maze by performing the pledge algorithm provided that there is such a solution. An example of the robot's path using an error-free pledge algorithm is given in Fig. 1. The angle counting technique is illustrated for the second obstacle: After the robot hits the obstacle in p_2 it turns about $-\frac{\pi}{2}$ to follow the wall. In p_3 the robot turns about $+\frac{\pi}{4}$ to follow the next wall. Finally, in p_4 it turns about $+\frac{\pi}{4}$ again until the angle counter reaches zero and the robot leaves the obstacle. Observe that the robot does not leave the obstacle in p_1, since its angle counter is -2π instead of zero.

In the following we define a class $\mathcal{K}$ of curves in the robot's workspace. The curves in $\mathcal{K}$ represent possible paths that lead to an exit, even if the robot's sensors and motions are erroneous. For convenience, we assume that the robot is point-shaped, but this is no restriction as long as the robot is smaller than the distances between obstacles. So the parts of a curve that map to a movement along a wall, are line segments on the boundaries of obstacles. To escape from an unknown maze, the robot's strategy is not required to calculate a path

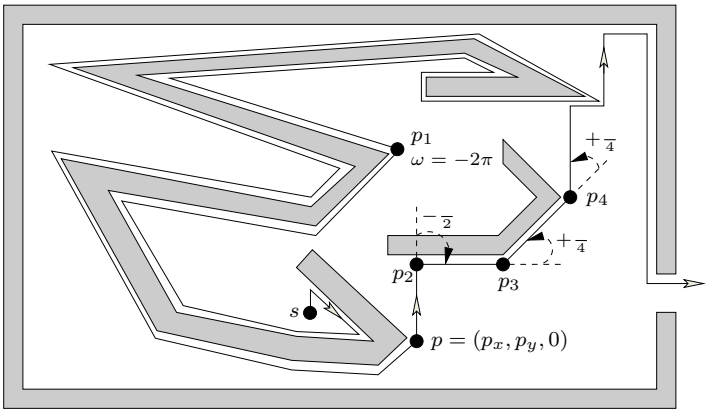


Fig. 1. The path of the pledge algorithm.

that matches a curve in $\mathcal{K}$ exactly, rather it is sufficient that the robot's path orientates essentially at a curve in $\mathcal{K}$. For example the robot may follow a wall in a certain distance, because it is not point-shaped, or it may follow a wall in a zig-zag manner. See Sect. 4 for more details.

Since every point on the curve represents a position as well as a heading, the curve is a subspace of the workspace $\mathcal{C} = \mathbb{R} \times \mathbb{R} \times \mathbb{R}$ and a point will be described as $C(t) = (P(t), \varphi(t))$, where $P(t) = (X(t), Y(t))$ denotes the position at time t and $\varphi(t)$ the *heading*. Note, that $\varphi(t)$ is the sum of all turns the curve has made so far, so if the curve has made two full turns counterclockwise then $\varphi(t)$ equals 4π instead of zero.

In order to classify the possible positions in the workspace, we divide the space of positions, $\mathcal{P} = \mathbb{R} \times \mathbb{R}$, into three subspaces: First, the space of forbidden configurations, $\mathcal{C}_{\text{forb}}$, the union of the interior of all obstacles. Second, the space of half-free configurations, $\mathcal{C}_{\text{half}}$, that is the union of the boundaries of the obstacles, and last the free configurations, $\mathcal{C}_{\text{free}}$, where $P(t) \notin P_i$ for all obstacles P_i holds.

If a curve hits a point $p = P(H_i)$ in $\mathcal{C}_{\text{half}}$ after a movement through $\mathcal{C}_{\text{free}}$, we will call H_i a *hit point*. If the curve leaves $\mathcal{C}_{\text{half}}$ and enters the free space at $q = P(L_i)$, we call L_i a *leave point*. With respect to the pledge algorithm we assume that every leave point belongs to a vertex of an obstacle.

3 Sufficient Conditions

Let us assume that it is possible to leave a given unknown maze. Within this section we establish a set of sufficient conditions for escaping from the maze. The robot follows—somehow—a curve C and to ensure that the robot can escape we want to avoid infinite cycles in the curve.

There are mainly two sources of errors in the pledge algorithm: one concerns the motion in the free space and the other the motion along the boundaries of the obstacles. The first type of error occurs, if the robot is not able to move a

straight line in the free space, the latter, if the robot's angle counter is in some way inaccurate. Both types lead to a set of sufficient conditions that ensures the correctness of an error-afflicted pledge algorithm.

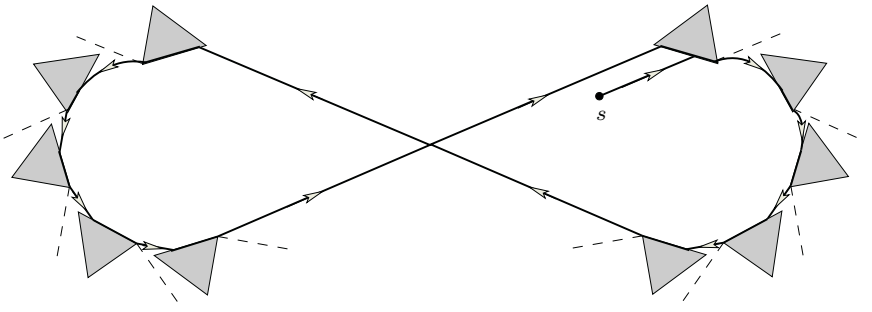


Fig. 2. Small errors along each boundary can sum up to a cycle.

To establish the set of conditions, we first observe that already small counting errors along the boundary of obstacles or deviations in the free space can sum up to a big mistake and lead to an infinite cycle. Figure 2 shows an example: The dashed lines show the correct leaving direction with respect to the direction in the hit point. Between the obstacles the robot drifts a little bit away from the correct direction and ends up in an infinite loop. This would happen even if the curve would not be allowed to make a full 2π turn in the free space. Obviously, cycles would be inhibited, if the curve between two obstacles would stay in a wedge around the initial direction. In fact, this wedge can be as large as a half space! This leads to the condition that for any two points in the free space, the difference in the headings should be lower than π. We will refer to this as the *free space condition.*

Unfortunately the free space condition is not sufficient. Figure 3 shows two examples, where C has a cycle, although the free space condition is fulfilled. In both cases the curve starts in s, meets an obstacle in H_i, misses the first possible leave point p and leaves the obstacle at another vertex q. The curve in Fig. 3(i) hits the same obstacle again in H_k. In Fig. 3(ii) the curve hits another obstacle, leaves this one in L_k and hits the first obstacle again in H_k. In both cases $P(H_k)$ is visited two times, at H_k and t_k. In the first case, the heading in t_k is slightly larger than π, in the second case $+\frac{\pi}{2}$. Observe that the curve in Fig. 3(ii) represents a second mistake: the heading in the hit point H_k is not zero, as it should be according to the pledge algorithm. This may occur if the last obstacle was left too early or the path between the obstacles is not a straight line segment or this may be a combination of both reasons. However, the heading in H_k is $-\frac{\pi}{2} - \varepsilon$ and both mistakes sum up to an error that is slightly larger than π, too.

Note, that the problem is not related to a second visit of a single point. The curve of the error-free pledge algorithm, may have many self-hits. It is rather

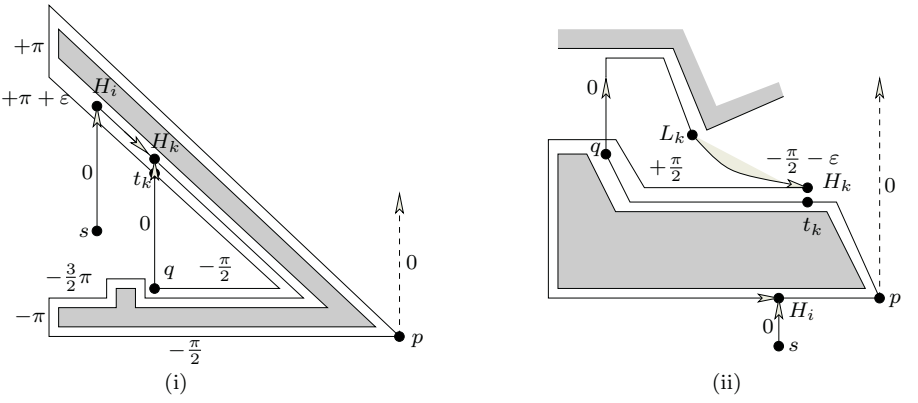


Fig. 3. Missing a leave point can lead to a cycle.

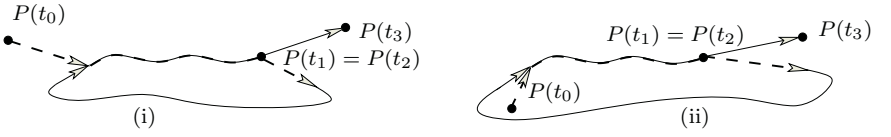


Fig. 4. The difference between a crossing (i) and a touch (ii) at t_2.

the fact that the heading in t_k is quasi overwinded in terms of the heading in the hit point.

Those observations lead to the conjecture that whenever the curve hits an obstacle at H_k and there exists a point t_k with $P(H_k) = P(t_k)$, then $\varphi(t_k) - \varphi(H_k) < \pi$ should hold. We will refer to this as the *obstacle condition*. In the rest of this chapter, we will show that both conditions are sufficient.

Definition 1. *Let $\mathcal{K}$ be the class of curves in $\mathcal{C}_{free} \cup \mathcal{C}_{half}$ that satisfy the following conditions:*

(i) The curve circles an obstacle in a counter-clockwise direction.
(ii) Every leave point belongs to a vertex of an obstacle.
(iii) $\forall t_1, t_2 \in C : P(t_1), P(t_2) \in \mathcal{C}_{free} \Rightarrow |\varphi(t_1) - \varphi(t_2)| < \pi$ *(Free space condition).*
(iv) $\forall H_i, t \in C : P(t) = P(H_i) \Rightarrow \varphi(t) - \varphi(H_i) < \pi$ (Obstacle condition).

Obviously, the curve of the error-free pledge algorithm is a curve in $\mathcal{K}$. The curves in $\mathcal{K}$ have two important properties that will be shown in the following two lemmata.

Lemma 2. *A curve $C \in \mathcal{K}$ cannot cross itself.*

Note, that a curve of $\mathcal{K}$ can touch itself, see Fig. 4.

Proof. Let us assume, C crosses itself. Consider the *first* crossing of C, i. e., there are two parameters t_1 and t_2 with $t_1 < t_2$ and $P(t_1) = P(t_2)$, where a crossing

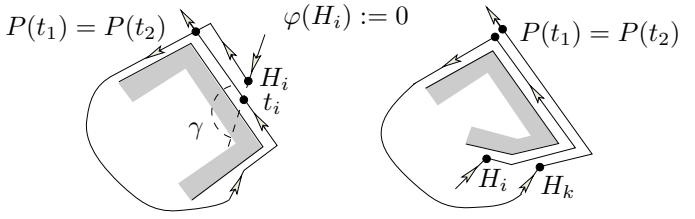


Fig. 5. A counterclockwise turn and a crossing respectively no crossing.

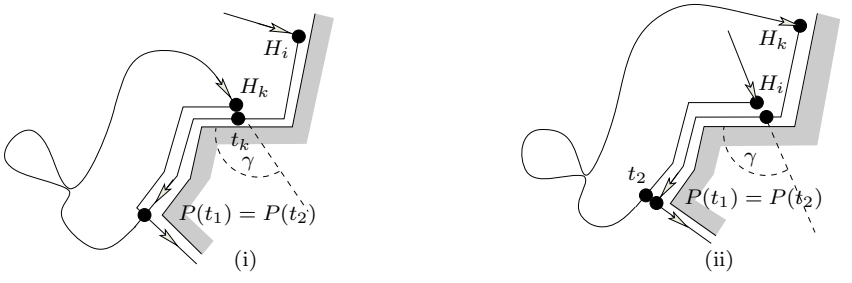


Fig. 6. A clockwise turn and a crossing respectively no crossing.

occurs in $P(t_2)$ and no crossings exist before t_2. The curve C will make either a counterclockwise or a clockwise loop between t_1 and t_2. If this first crossing happens in the free space, the curve will violate the free space condition, so the first crossing could occur only in $\mathcal{C}_{\text{half}}$.

Let us consider the case of a counterclockwise loop, see Fig. 5(i). The curve hits an obstacle at H_i, makes a counterclockwise turn, meets $P(H_i)$ for t_i again and has a crossing for $t_2 > t_i > H_i$. Note, that there is no crossing, if the point $P(H_i)$ is not met between t_1 and t_2, see Fig. 5(ii).

W. l. o. g. we assume $\varphi(H_i) = 0$. The loop may leave the obstacle or not, however, we reach t_i with the heading $\varphi(t_i) + (-\gamma) = 2\pi$. At $P(H_i)$ the robot turns clockwise with angle γ to follow the obstacle boundary, so $-\pi < \gamma < 0$ must hold. Hence $\varphi(t_i)$ is greater than π and the obstacle condition is violated.

Now we look at a clockwise turn. The curve hits an obstacle at H_i, follows the obstacle and leaves the obstacle. Eventually, it returns to the obstacle at another hit point $H_k > H_i$ and has a crossing at t_2, see Fig. 6(i). The point $P(H_k)$ has to be met before at t_k between H_i and t_1. Otherwise the curve only touches itself and there is no crossing at t_2, see Fig. 6(ii).

Let $\varphi(H_k^+)$ denote the heading immediately after the robot has turned in H_k, thus $\varphi(H_k^+) = \varphi(H_k) + \gamma$. As above we have $-\pi < \gamma < 0$. On the other hand, the curve has made a full clockwise turn between t_k and H_k^+, therefore $\varphi(H_k^+) = \varphi(t_k) - 2\pi$. Finally, the obstacle condition has to be fulfilled, too:

$$\begin{aligned} & \varphi(t_k) - \varphi(H_k) < \pi \\ \Leftrightarrow\ & \varphi(H_k^+) + 2\pi - \varphi(H_k) = \varphi(H_k) + \gamma + 2\pi - \varphi(H_k) < \pi \\ \Leftrightarrow\ & \gamma < -\pi \end{aligned}$$

It follows that the first crossing cannot exist, and—by induction—the curve cannot cross itself. □

Lemma 3. *A curve $C \in \mathcal{K}$ will hit every edge in the environment at most once.*

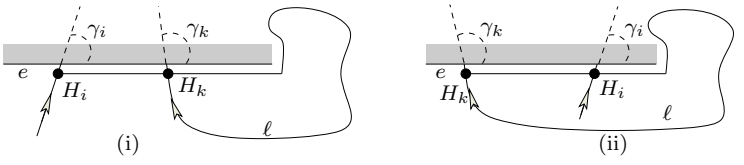


Fig. 7. A curve that hits an edge twice.

Proof. Let us assume, the curve C will hit an edge e more than once: after a first hit at H_i the robot moves around and hits e again at H_k, see Fig. 7. At $P(H_i)$ respectively $P(H_k)$ the robot turns clockwise to follow the edge e, therefore $-\pi < \gamma_i, \gamma_k < 0$. Let $\varphi(H_i^+)$ and $\varphi(H_k^+)$ be defined as in the proof of Lemma 2. W. l. o. g. we assume $\varphi(H_i^+) = 0$. Since the curve in H_i^+ and H_k^+ follows the same edge e, the headings $\varphi(H_i^+)$ and $\varphi(H_k^+)$ must be mod 2π the same, i. e., $\varphi(H_k^+) = 2k\pi, k \in \mathbb{Z}$. For $k \neq 0$, with $\varphi(H_i) = -\gamma_i$ and $\varphi(H_k) = \varphi(H_k^+) - \gamma_k$ it follows that $|\varphi(H_k) - \varphi(H_i)| = |2k\pi - \gamma_k + \gamma_i| > \pi$ and the free space condition would be violated.

Therefore, we can assume $k = 0$ and $\varphi(H_k^+) = 0$. Consider the part of C between the first and the second visit of $P(H_k)$ (in a situation as shown in Fig. 7(i)) respectively the curve between the consecutive visits of $P(H_i)$ (in a situation as shown in Fig. 7(ii)). If this loop, ℓ, has no crossings, then ℓ is a jordan curve and C makes a $\pm 2\pi$ turn in ℓ. Thus $\varphi(H_k^+)$ equals $\pm 2\pi$, in contradiction to our assumption. Hence the curve between the two visits of e must have at least one crossing. But this contradicts to Lemma 2. □

Finally, with Lemma 2 and Lemma 3 we are able to show that the conditions from Definition 1 are sufficient to solve our problem.

Theorem 4. *A robot, whose path follows a curve $C \in \mathcal{K}$, will escape from an unknown maze, if this is possible at all.*

Proof. A curve that meets the conditions from Definition 1 will hit every edge in the environment at most once. Therefore the robot will either escape at the latest after it has visited all edges, or it will be trapped, because there is no escape from the maze. □

4 Applications

Within this section we discuss the practical relevance of Theorem 4. What consequences does it have for the design of a robot that should be able to leave an unknown maze?

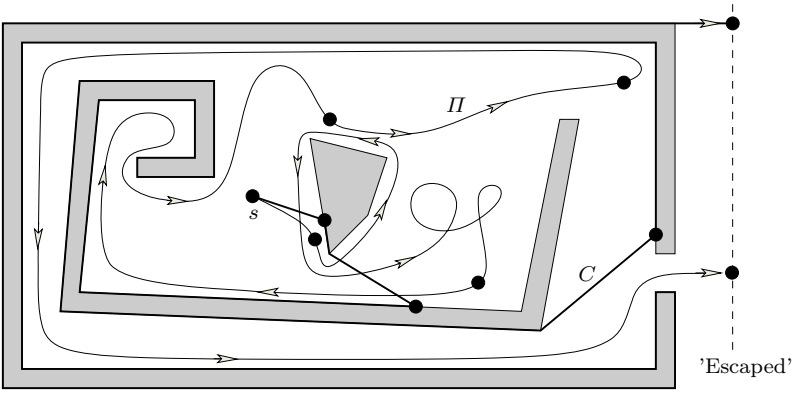


Fig. 8. A curve and a related robot's path.

Still we have to discuss, what it means that the robot follows a curve $C \in \mathcal{K}$. In fact, the deviation from the robot's path, Π, and C can be as large as shown in Fig. 8. The path Π might be produced by an arbitrary strategy, C shows the corresponding curve. The dots on Π represent the hit points in terms of the robot's strategy, i. e., the strategy switches from a move-through-free-space behavior to a go-around-obstacle behavior. Amazingly it is sufficient that the hit points met by C and Π refer to the same edges—including an imaginary 'final edge' which marks the successful escape from the maze. The sequence of hit points is a kind of Ariadne's Thread that leads the robot out of the maze. Between the hit points the robot can do whatever it wants, it can even hit some other edges.

More precisely let $\mathcal{H}_C$ be the sequence of edges hit by the curve C, and $\mathcal{H}_\Pi$ the sequence of edges that are related to a hit point of the robot, then the following holds:

Corollary 5. *A robot escapes from an unkown maze, if there exists a curve $C \in \mathcal{K}$, such that $\mathcal{H}_C$ is a subsequence of $\mathcal{H}_\Pi$.*

4.1 A Simple Compass Is Sufficient

Let us now assume that the robot is equipped with an error afflicted compass. Additionally, the robot should be able to follow walls and walk in the free space until it *detects* hit points. Let the robot measure its turning angles by using compass values. How this is implemented will depend on the specific robot. Note, that just (even exact) compass readings without measuring turning angles is not sufficient, because only directions mod 2π can be detected in this case.

If the compass has a measuring error less than $\frac{\pi}{2}$ it should be easy to ensure that the heading of the robot in the free space remains in an intervall of $]-\frac{\pi}{2}, \frac{\pi}{2}[$. This guarantees the free space condition at hand. Additionally, at every *detected* hit point H_i we have $\varphi(H_i) \in]-\frac{\pi}{2}, \frac{\pi}{2}[$.

On the other side it can be assumed that it is easy to detect 2π or -2π turns along the walls, although again error afflicted within a range of $]-\frac{\pi}{2}, \frac{\pi}{2}[$, due to the compass inaccuracy. Therefore we can assume that the total measured turning angle along the walls is always within a bound of $]-\frac{\pi}{2}, \frac{\pi}{2}[$ away from the exact turning angle. So while the robot follows a wall being at position $t \in C$ we have $\varphi(t) < \frac{\pi}{2}$. Altogether, the obstacle condition is fulfilled.

Corollary 6. *Let an unknown maze be given and let the robot be equipped with an error afflicted compass. Let us assume that it is possible to escape from the maze. If the accuracy of the compass is not worse than $\frac{\pi}{2}$, a simple pledge-like strategy leads to an exit of the unknown maze.*

4.2 Pseudo Rectilinear Scenes

Now we assume that the environment has more or less axis-parallel walls. To be more precise we introduce the notion of (α, β) *pseudo-rectilinear polygons* and specify the robot's ability with respect to α and β.

A *convex (*concave*) corner* of a simple polygon P is a vertex of P with outer angle greater (smaller) than π. A simple polygon P is called *pseudo-rectilinear* iff $|\{p \,|\, p \text{ is a convex corner of } P\}| = |\{p \,|\, p \text{ is a concave corner of } P\}| + 4$.

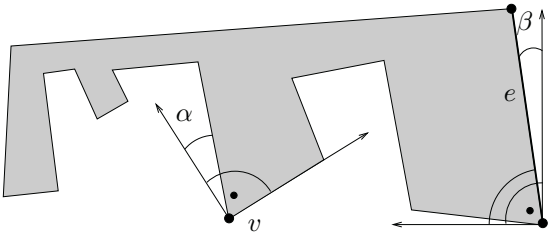


Fig. 9. A pseudo-rectilinear polygon with $\text{anglediv}(v) = \alpha$ and $\text{edgediv}(e) = \beta$.

In Fig. 9 there is an example of a pseudo-rectilinear polygon. There are several ways to measure the quality of rectilinearity of a pseudo-rectilinear polygon. Our aim is that the robot should be able to count the turns along obstacles by considering the number of passed convex and concave vertices. Therefore it is important to detect whether a vertex is convex or not which gives rise to the following definitions.

For a vertex v let $\angle(v)$ denote the outer angle at v. We define the *angle-divergence* at a single corner as follows (see Fig. 9):

$$\text{anglediv}(v) := \begin{cases} \left|\angle(v) - \frac{\pi}{2}\right| & : v \text{ is a concave corner} \\ \left|\angle(v) - \frac{3\pi}{2}\right| & : v \text{ is a convex corner} \end{cases}$$

For a pseudo-rectilinear polygon P we define the angle-divergence $\text{angledir}(P)$ of P by

$$\text{angledir}(P) := \max_{v \in P} \text{angledir}(v)\,.$$

On the other hand the robot should be able to leave an obstacle along a wall in 'horizontal' or 'vertical' direction. For this purpose we have to consider the divergence of the edges from the X-axis respectively Y-axis.

For an edge $e = vw$ let $\text{edgediv}(e)$ denote the smallest angle between e and an axis-parallel line through v or w (see Fig. 9). For a pseudo-rectilinear polygon P we define the *edge-divergence* $\text{edgediv}(P)$ of P by

$$\text{edgediv}(P) := \max_{e \in P} \text{edgediv}(e)\,.$$

A pseudo-rectilinear polygon P is called (α, β) *pseudo-rectilinear* iff $\text{angledir}(P) \leq \alpha$ and $\text{edgediv}(P) \leq \beta$ holds.

We assume that the robot measures angles at corners respectively between its heading and a wall within an accuracy of δ, that is, the absolute difference of the actual angle and the measured angle is smaller than δ. In order to guarantee that convex and concave corners are detected correctly it obviously suffices to require that $\alpha + \delta < \frac{\pi}{2}$, see Fig. 10.

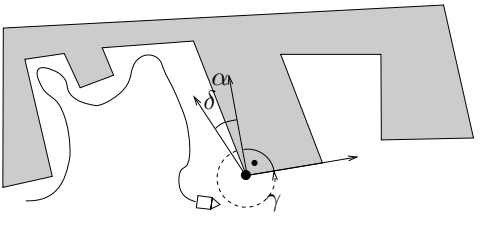


Fig. 10. Detecting concave and convex corners under the presence of errors α and δ. Obviously, $\alpha + \delta$ should not exceed $\frac{\pi}{2}$.

W. l. o. g. we assume that the robot's predefined direction is north, i. e., $\frac{\pi}{2}$ in the mathematical sense. The correctness of the movement in the free space will depend on β and δ, we have to require that the next hit point is appropriate. First, we require that the robot's path between a leave point and a hit point should be monotone with respect to the leaving direction. On the other hand it is important that the robot hits a horizontal edge at the end of its free space movement, thus it will be able to go on counting correctly. Note, that between two hit points the robot can hit and detect vertical walls. In this case we let the robot simply slide along them.

We want to make sure that the robot is able to detect whether an edge is 'horizontal' or 'vertical'. Naturally, if the measured angle between the robot's heading and an edge is smaller than $\frac{3}{4}\pi$ and greater than $\frac{\pi}{4}$, it is reasonable that the robot assumes that the edge is horizontal.

In this sense it suffices to guarantee that the robots heading in the free space lies in the range $\left]\frac{\pi}{4}+\beta+\delta, \frac{3}{4}\pi-\beta-\delta\right[$. For example in Fig. 11 the robot leaves the obstacle at point p and meets edge e with heading ϕ. Since e should be detected as 'vertical' the angle $\gamma-\delta$ should be greater than $\frac{3}{4}\pi$. We have $\gamma=\pi-\left(\pi-\left(\frac{\pi}{2}-\beta\right)-\phi\right)$ and therefore we require $(\gamma-\delta)=\frac{\pi}{2}-(\beta+\delta)+\phi>\frac{3}{4}\pi$ which in turn is equivalent to $\phi>\frac{\pi}{4}+\beta+\delta$. With similar arguments we get the upper bound of the range $\left]\frac{\pi}{4}+\beta+\delta, \frac{3}{4}\pi-\beta-\delta\right[$.

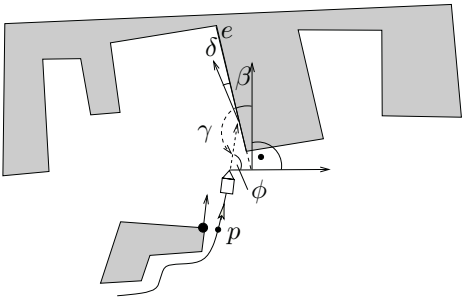


Fig. 11. Detecting 'horizontal' and 'vertical' edges under the presence of errors β and δ.

We assume that the robot counts convex and concave corners rather than counting angles at obstacles. With respect to the leaving direction at a leave point the robot always can start with a heading in the range $\left[\frac{\pi}{2}-\beta, \frac{\pi}{2}+\beta\right]$. Altogether, according to the range above, it suffices to require that the robot should be able to guarantee this heading up to an angle $\frac{\pi}{4}-\delta-2\beta\geq 0$. Obviously, $2\beta\geq\alpha$ holds. Therefore the condition $\alpha+\delta<\frac{\pi}{2}$ is already fulfilled, if $\frac{\pi}{4}-\delta-2\beta\geq 0$ holds.

Corollary 7. *Let an unknown maze with a set of (α,β) pseudo-rectilinear polygons be given. Let the robot be able to measure angles within an accuracy of δ. Let us assume that it is possible to escape from the maze.*

If the robot is able to guarantee its heading in the free space up to an angle $\frac{\pi}{4}-\delta-2\beta$, a simple pledge-like strategy lead to an exit of the unknown maze.

4.3 Exact Free Motion

Now let us assume that the robot is able to move a straight path between obstacles, just its angle counter is inaccurate in some way. Let β_i denote the difference between the real angle and the measured angle at the ith turn and n the number of vertices in the environment.

Lemma 8. *If the robot is able to move a straight path in the free space and ensures that $\left|\sum_{i=k}^{\ell}\beta_i\right|<\pi$ holds for all $k\leq\ell\leq m$, where m denotes the number*

of turns the robot has made so far, then it will be able to escape from an unknown maze.

Proof. The new condition implies that the absolute value of the measuring error that is the difference between the robot's heading and its angle counter, never exceeds π. Now we have to show that our new condition does not violate the conditions from Definition 1.

First, let us assume that the free space condition is not met, so there must be two points t_1, t_2 in the free space where $|\varphi(t_1) - \varphi(t_2)| \geq \pi$ holds. W. l. o. g. we define $\varphi(t_1) = 0$. Since the robot moves a straight line in the free space, the headings at the leave point, the following hit point and all points between them must be the same. So there must be a leave point L_k with $|\varphi(L_k)| = |\varphi(t_2)| \geq \pi$. But the robot leaves an obstacle only, if its angle counter has reached zero, so the absolute value of the measuring error in L_k is at least π.

Second we assume, the obstacle condition would be violated. Then there must be an obstacle P_i with a hit point H_k and another point t_k with $P(t_k) = P(H_k)$, such that $\varphi(t_k) - \varphi(H_k) > \pi$ holds. Let w. l. o. g. $\varphi(H_k) = 0$, then $\varphi(t_k) \geq \pi$ holds, but the angle counter can't be greater then zero, since the robot leaves the obstacle as soon as the counter becomes zero. So the difference between the robot's heading and its angle counter must be at least π. □

Now, let $\beta_{\max} := \max \beta_i$. With Lemma 3 it is easy to see that the robot visits at most n^2 vertices, and we have the following result:

Corollary 9. *A robot that guarantees* $|\beta_{\max}| < \frac{\pi}{n^2}$ *is able to escape.*

5 Conclusion

We considered the pledge algorithm under errors in sensors and motion and established sufficient requirements for the robot for leaving an unknown maze. Especially, we showed, that a robot equipped with a simple compass will fulfill this task, we gave upper bounds for errors in pseudo rectilinear mazes and for robots with correct motion. Furthermore, in Corollary 5 we gave a framework for proving the correctness of arbitrary strategies for leaving mazes.

Within a simulation environment we have incorporated several sources of errors into an implementation of the pledge algorithm, see [6]. Next, we are going to implement the pledge algorithm for a Pioneer 2 AT Robot and will further observe its behavior on pseudo rectilinear mazes and other error afflicted settings. We are quite sure that on smooth terrains our robot will be able to fulfill the required conditions.

Acknowledgement

We would like to thank Peter Gritzmann for pointing out the problem and Rolf Klein for helpful discussions.

References

1. H. Abelson and A. A. diSessa. *Turtle Geometry*. MIT Press, Cambridge, 1980.
2. M. A. Batalin and G. S. Sukhatme. Efficient exploration without localization. In *Proc. IEEE Internat. Conf. Robot. Autom.*, 2003.
3. P. Berman. On-line searching and navigation. In A. Fiat and G. Woeginger, editors, *Competitive Analysis of Algorithms*. Springer-Verlag, 1998.
4. X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment I: The rectilinear case. *J. ACM*, 45(2):215–245, 1998.
5. P. Gritzmann. Personal communication via Rolf Klein.
6. U. Handel, T. Kamphans, E. Langetepe, and W. Meiswinkel. Polyrobot - an environment for simulating strategies for robot navigation in polygonal scenes. Java Applet, 2002. http://web.informatik.uni-bonn.de/I/GeomLab/Polyrobot/.
7. A. Hemmerling. *Labyrinth Problems: Labyrinth-Searching Abilities of Automata*. B. G. Teubner, Leipzig, 1989.
8. F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM J. Comput.*, 31:577–600, 2001.
9. C. Icking, T. Kamphans, R. Klein, and E. Langetepe. On the competitive complexity of navigation tasks. In H. B. Hendrik I. Christensen, Gregroy D. Hager and R. Klein, editors, *Sensor Based Intelligent Robots*, volume 2238 of *LNCS*, pages 245–258, Berlin, 2002. Springer.
10. I. Kamon and E. Rivlin. Sensory-based motion planning with global proofs. In *Proc. 13th IEEE Internat. Conf. on Robotics and Automation*, pages 814–822, 1997.
11. S. L. Laubach. *Theory and Experiments in Autonomous Sensor-Based Motion Planning with Applications for Flight Planetary Microrovers*. Ph.D. thesis, California Institute of Technology, Pasadena, CA, 1999.
12. V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.
13. J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
14. S. Rajko and S. M. La Valle. A pursuit-evasion bug algorithm. In *Proc. IEEE Conf. Robotics Automat.*, 2001.
15. N. S. V. Rao, S. Kareti, W. Shi, and S. S. Iyengar. Robot navigation in unknown terrains: introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, 1993.
16. M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 40, pages 733–754. CRC Press LLC, Boca Raton, FL, 1997.
17. B. Yamauchi, A. Schultz, and W. Adams. Mobile robot exploration and map-building with continuous localization. In *Proc. IEEE Internat. Conf. Robot. Autom.*, 1998.

The Online Matching Problem on a Line*

Elias Koutsoupias[1,2] and Akash Nanavati[2]

[1] Department of Informatics, University of Athens,
Panepistimiopolis, Athens 15784, Greece
elias@di.uoa.gr
[2] Computer Science Department, University of California Los Angeles,
Los Angeles, CA 90095, USA
akash@cs.ucla.edu

Abstract. We study the *online matching* problem when the metric space is a single straight line. For this case, the offline matching problem is trivial but the online problem has been open and the best known competitive ratio was the trivial $\Theta(n)$ where n is the number of requests. It was conjectured that the generalized *Work Function Algorithm* has constant competitive ratio for this problem. We show that it is in fact $\Omega(\log n)$ and $O(n)$, and make some progress towards proving a better upper bound by establishing some structural properties of the solutions. Our technique for the upper bound doesn't use a potential function but it reallocates the online cost in a way that the comparison with the offline cost becomes more direct.

1 Introduction

We study the online version of the weighted bipartite matching problem in which the nodes of one partition, called the *servers*, are known in advance and the nodes of the other partition, the *requests*, arrive online and must be matched immediately. The objective is to minimize the cost of the matching. The more general problem in which the nodes of both partitions arrive online and must be matched as soon as possible is not essentially different; by appropriate partitioning the sequence of points, it is simply a repeated version of the simple version and all our results apply to the more general problem. Of special interest is the metric matching problem in which the points, both servers and requests, lie in some metric space and especially in a Euclidean space.

The offline matching problem is one of the most fundamental algorithmic problems and it has played a central role in the development of the theory of algorithms [2, 3, 10]. The online version of the problem has also been studied before.

Karp, U. Vazirani, and V. Vazirani [8] were the first to study some online version of the matching problem. They studied randomized algorithms for online matching in an unweighted bipartite graph when the optimum matching was a perfect matching. Khuller, Mitchell, and Vazirani [9] studied the online version

* Research supported in part by NSF.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 179–191, 2004.
© Springer-Verlag Berlin Heidelberg 2004

of minimum bipartite matching. They showed that for non-metric spaces the competitive ratio is unbounded. They also proposed a natural algorithm, *Permutation*, which has optimal competitive ratio for arbitrary metric spaces; its competitive ratio is $2n - 1$, where n is the number of requests. The same algorithm was studied independently by Kalyanasundaram and Pruhs [5, 7]. They conjectured that the Work Function Algorithm has constant competitive ratio (see their review article [6]).

Unlike the minimization version of the online matching the maximization version has been completely settled. For non-metric spaces the competitive ratio is unbounded [9] and for metric spaces the greedy algorithm is optimal and has competitive ratio 3 [5].

Here we are concerned with the special case of the metric online minimum matching where the servers and requests lie on a line (1-dimensional Euclidean space). This is perhaps the most interesting case of online matching for many reasons. First, the offline matching problem for this case is trivial and therefore competitive analysis appropriately captures the deterioration of performance (that is, the approximation ratio or competitive ratio) due to lack of complete information. Second, it is a natural generalization of other fundamental online problems such as the cow-path problem (also known as the bridge problem) [1]. Third, versions of online matching problem play a central role in electronic markets where "buyers" and "sellers" arrive online, name their prices, and must be "matched". One can view the prices as points on a line that must be matched online. The version of the matching problem that we study in this work together with the analogous maximization version, are the simplest, pure abstractions of these online matching problems.

It is easy to show that no online algorithm can have competitive ratio less than 9 since the online matching problem is a generalization of the cow-path problem (see the beginning of the Section 3). Very recently Fuchs, Hochstättler, and Kern [4] gave a lower bound of 9.001 which suggests that the problem is not simply a disguised generalization of the cow-path problem. It has been conjectured [6] that the matching problem on the line and in particular the Work Function Algorithm (WFA) have constant competitive ratio. Here we disprove the conjecture and establish a lower bound of $\Omega(\log n)$ where n is the number of requests. We also show an upper bound $O(n)$ for the competitive ratio of WFA. The bound by itself is weak but the proof technique is more important. It exhibits structural properties of the solutions that can lead to improved bounds. Using these properties we show that the competitive ratio is no more that the height of the online matching (i.e., the maximum number of lines through a point). We can also establish a tight upper bound ($O(\log n)$) for some special cases (restricted adversaries), but due to lack of space we omit these results from this abstract.

Outline

In Section 2 we define the WFA and prove some properties that are useful in general and in particular they simplify the argument for the lower bound. We then give the relatively simple lower bound (Section 3).

To prove the upper bound, we first establish some useful properties and invariants of the WFA (beginning of Section 4). We now want to compare the cost of the WFA to the optimal matching (*opt*). However, the optimal matching can use different set of servers than WFA. We show (Subsection 4.1) that this is not a serious problem: the WFA uses an almost optimal set of servers in the sense that the optimal offline matching involving the servers of WFA (*popt*) is within a constant factor from the optimal (*opt*). We need therefore only to compare the online cost with *popt*. We don't know how to do it directly though. As it is frequently the case, it is not easy to apply the potential function method to online algorithms with non-constant competitive ratio. Indeed, we don't use a potential function but we find a way to reallocate the online cost in such a way that it becomes easier to compare it to *popt*. A major obstacle is that *popt* is not monotone: more requests can decrease the matching. Furthermore, new requests may decrease the number of *popt* lines that cross a particular point (see β_x in the Section 2). We get around these problems by introducing a monotone quantity, the alive parts of the offline matchings (*alive*), which counts each interval with multiplicity roughly equal to the maximum number of *popt* lines that crossed it in the past. We then show that *alive* is also within a constant factor from *popt* (Subsection 4.2). Finally, it is easy to bound the online cost by *alive*, and thus indirectly with the optimum (Subsection 4.3).

We conclude and mention some open problem in Section 5.

2 Properties of the Work Function Algorithm

The metric online matching problem is defined on a metric space $\mathcal{M}$ endowed with distance d. A (possibly infinite) subset of $S \subseteq \mathcal{M}$ is the set of servers. Some of these servers are going to be matched with a sequence of requests $\{r_1, r_2, \ldots\} \subseteq \mathcal{M}$ [1]. Let $R_n = \{r_1, \ldots, r_n\}$ denote the set of the first n requests and let $S_n = \{s_{r_1}, \ldots, s_r\}$ denote the set of servers used by the online algorithm to match these requests.

For the problem we study here the metric space is the 1-dimensional Euclidean space. Let $M(A, R)$ denote the optimal (minimum) matching between sets A and R with $|A| = |R|$. Clearly $M(A, R)$ can be obtained by matching the leftmost request to the leftmost server and recursing in the same fashion. There may be other optimal matchings but we will use the notation $M(A, R)$ for this particular matching. For simplicity, we also denote the weight of the optimal matching by $M(A, R)$.

Given the definition of the matching $M(A, R)$ we define

$$\beta_x(A, R) = |\{a : a \in A \text{ and } a < x\}| - |\{r : r \in R \text{ and } r < x\}|,$$

which is the number of lines in $M(A, R)$ that cross point x with appropriate sign.

[1] Here we assume that all servers and requests are distinct points. It is obvious that the general case of allowing multiple servers and/or requests on the same point is the limit case when some distances tend to zero. For simplicity we sometimes allow multiple servers/requests on points though. Also, all sets are assumed to be multisets.

We denote singletons $\{a\}$ by their unique element a, and the multiset consisting of k copies of a by a^k. In particular, $B + a^k$ will denote the multiset that results if we add k copies of a to multiset B.

Definition 1 (Work Function Algorithm). *The* Generalized Work Function Algorithm *(γWFA) matches request r_n to an unmatched server $s_r \in S - S_{n-1}$ that minimizes the expression: $\gamma M(S_{n-1} + s_r, R_{n-1} + r_n) + d(s_r, r_n)$.*

We state the following easy result without proof.

Proposition 1. *For $\gamma = 0$ this is the greedy algorithm with competitive ratio at least $2^n - 1$, and in general, for $0 \le \gamma < 1$, competitive ratio is exponential, at least $(\zeta^n - 1)/(\zeta - 1)$, where $\zeta = 2/(\gamma + 1)$. The case $\gamma = 1$ is known simply as the Work Function Algorithm, with competitive ratio at least n. The case $\gamma = \infty$ (i.e. ignore $d(x, r)$ and choose s_r that minimizes $M(S_{n-1} + s_r, R_{n-1} + r_n)$) is the retrospective algorithm also known as the* Permutation *algorithm [5] with competitive ratio at least n (and at most $2n - 1$).*

Our analysis applies to any $\gamma > 1$. There are indications that the minimum competitive ratio is achieved for approximately $\gamma = 3$; in particular for the special case of the cow-path problem the value $\gamma = 3$ is optimal. We decided to state our result for general γ instead of $\gamma = 3$ because this slight generalization is sometimes helpful to the reader (by reducing the number of "magic constants").

Let r be a request and let s_1 be the rightmost unmatched server in the interval $(-\infty, r)$. Similarly let s_2 be the leftmost unmatched server in (r, ∞). We call the two servers s_1 and s_2 the *surrounding* servers of r. It is easy to see that any online algorithm can be converted into one that services each request with one of its two surrounding servers with the same (or better) competitive ratio. Our first aim is to show that the γWFA has this nice property:

Property 1 (Locality). The γWFA services each request with one of its two surrounding servers.

To show the Locality Property we need to take into account the history of the matching created by the γWFA. To see this consider the following example: There are three servers s_1, s_2, s_3 (Figure 1).

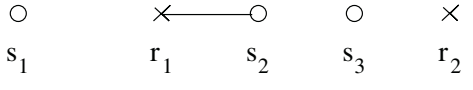


Fig. 1. Surrounding servers

The first request r_1 is in the interval (s_1, s_2) and is matched to s_2. The second request is in (s_3, ∞). If γWFA has Property 1, then it must match r_2 to s_3. This can be easily verified, but only if we take into account the history of γWFA and in particular the fact that $d(s_1, r_1) \ge d(r_1, s_2)$. This example shows that we need

some kind of induction to show that γWFA has the Locality Property. It turns out that the Locality Property follows from the following invariant (in fact, it is equivalent to it).

Theorem 1. *Let s_1, s_2 be two servers that, before time t, have not been matched by the γWFA algorithm but every other server in (s_1, s_2) has been matched. Let A, R be the sets of servers and requests before time t in (s_1, s_2). Then the γWFA has matched the set of requests R to the same-cardinality set of servers A; furthermore*

$$\gamma M(A, R) \leq \gamma M(A + s_1, R + s_2) + d(s_1, s_2). \tag{1}$$

Before we proceed to prove the theorem, we point out that the Locality Property before time t follows immediately from it and in particular from the conclusion that A and R have the same cardinality and are matched by γWFA. Conversely, the theorem follows if we assume that γWFA has the Locality Property at time t: let the request r at time t be to the right of s_2 within very small distance; then the γWFA will prefer to match r to s_2 than to s_1 and this gives (1).

We will also need the following lemma about matchings:

Lemma 1. *Let s_1, s_2 be two servers and let A and R be two sets of points (servers and requests) in (s_1, s_2) of equal cardinality. Let also $r_1 \leq r_2$ be two points (requests) in $[s_1, s_2]$. Then*

$$M(A+s_1, R+r_1)+M(A+s_1, R+r_2) \leq M(A+s_1+s_1, R+r_1+r_2)+M(A, R). \tag{2}$$

Proof. Fix a point $x \in [s_1, r_1)$. The total number of lines that include x of the left-hand side is $|\beta_x(A+s_1, R+r_1)| + |\beta_x(A+s_1, R+r_2)| = 2|\beta_x(A, R)+1|$ and of the right-hand side is $|\beta_x(A+s_1+s_1, R+r_1+r_2)| + |\beta_x(A, R)| = |\beta_x(A, r)+2| + |\beta_x(A, r)|$. Since $2|a+1| \leq |a+2| + |a|$ (the absolute function is concave), it follows that the contribution of x to the left hand side of the inequality is no more than the right hand side. A similar situation holds when $x \in [r_1, r_2)$ and $x \in [r_2, s_2)$ and the lemma follows. □

Proof (of Theorem 1). To prove Theorem 1 we use induction on the number of requests in R and the properties of the γWFA. Let r be the most recent request in R and let s be the server matched to r by the γWFA. By induction, the Locality Property holds and therefore s is in the interval (s_1, s_2), that is $s \in A$. We consider two cases.

Case 1: $r < s$**.** Let A_1 and A_2 be the set of servers in (s_1, s) and (s, s_2), respectively. Let also R_1 and R_2 be the associated requests in these intervals (requests before r). By induction, A_1 (resp. A_2) has the same cardinality with R_1 (resp. R_2).

By the definition of γWFA we have $\gamma M(A_1+s, R_1+r)+d(s, r) \leq \gamma M(A_1+s_1, R_1+r)+d(s_1, r)$. By the induction hypothesis we also have $\gamma M(A_2, R_2) \leq \gamma M(A_2+s, R_2+s_2)+d(s, s_2)$. Therefore:

$$\begin{aligned}
\gamma M(A,R) &= \gamma M(A_1+s, R_1+r) + \gamma M(A_2, R_2) \\
&\le [\gamma M(A_1+s_1, R_1+r) + d(s_1,r) - d(s,r)] \\
&\quad +[\gamma M(A_2+s, R_2+s_2) + d(s,s_2)] \\
&= \gamma M(A_1+A_2+s_1+s, R_1+R_2+r+s_2) \\
&\quad +[d(s_1,r) - d(s,r) + d(s,s_2)] \\
&\le \gamma M(A_1+A_2+s_1+s, R_1+R_2+r+s_2) + d(s_1,s_2) \\
&= \gamma M(A+s_1, R+s_2) + d(s_1,s_2)
\end{aligned}$$

Case 2: $r \ge s$**.** Let again A_1, R_1 and A_2, R_2 be as in the first case. Then $M(A,R) = M(A_1+A_2+s, R_1+R_2+r) = M(A_1,R_1) + M(A_2+s, R_2+r)$.

By induction on the left interval, we have $\gamma M(A_1,R_1) \le \gamma M(A_1+s_1, R_1+s) + d(s_1,s)$. Also by induction on the right interval, we have $\gamma M(A_2,R_2) \le \gamma M(A_2+s, R_2+s_2) + d(s,s_2)$ or equivalently

$$\gamma M(A_2,R_2) - \gamma M(A_2+s, R_2+s_2) \le d(s,s_2).$$

But by Lemma 1 we have

$$M(A_2+s,R_2+r) - M(A_2+s+s, R_2+r+s_2) \le M(A_2,R_2) - M(A_2+s, R_2+s_2).$$

Combining the two inequalities we get $\gamma M(A_2+s,R_2+r) - \gamma M(A_2+s+s, R_2+r+s_2) \le d(s,s_2)$. Therefore:

$$\begin{aligned}
\gamma M(A,R) &= \gamma M(A_1,R_1) + \gamma M(A_2+s, R_2+r) \\
&\le [\gamma M(A_1+s_1, R_1+s) + d(s_1,s)] \\
&\quad +[\gamma M(A_2+s+s, R_2+r+s_2) + d(s,s_2)] \\
&= \gamma M(A_1+A_2+s+s_1, R_1+R_2+r+s_2) + d(s_1,s_2) \\
&= \gamma M(A+s_1, R+s_2) + d(s_1,s_2)
\end{aligned}$$

Notice the crucial use of Lemma 1. We remark that it we could get away without it when $A_1 \neq \emptyset$, but it is absolutely necessary otherwise. This is because in that case $|A_2+s| = |A|$ and so we cannot apply induction.

We also remark that in the second case we used only induction, no property of the γWFA at all. □

The following invariant is a generalization of Theorem 1 and we state it without proof:

Theorem 2. *Let s_1, s_2 be as in Theorem 1. Then for any points $v_1, \ldots, v_k$ with $s_1 \le v_1 \le v_2 \le \cdots \le v_k \le s_2$ we have*

$$\gamma M(A+s_1^k, R+v_1+\cdots+v_k) \le \gamma M(A+s_1^{k+1}, R+v_1+\cdots+v_k+s_2) + d(s_1,s_2), \quad (3)$$

where s_1^m denotes m copies of s_1.

Definition 2 (Balanced Interval). *Let s_1, s_2 be two unmatched servers such that inside (s_1,s_2) all servers are already matched. We will call such an interval* balanced.

Because of the Locality Property, we can focus our analysis to a balanced interval.

3 Lower Bound

We remark that the cow-path problem is a special case of the matching problem and this implies a lower bound of 9 for any online algorithm. In the reduction to the cow-path problem, there is a server on each integral point of R except 0. The first request is at $r_1 = 0$, and request r_t, $t > 1$, is at $s_{r_{t-1}}$, the server used to service the previous request.

We will now show a lower bound of $\Omega(\log n)$ on the competitive ratio of γWFA. This disproves a conjecture of Kalyanasundaram and Pruhs. We take advantage of the Locality Property of γWFA to simplify the argument.

Theorem 3. *The competitive ratio of* γWFA *is at least* $2 + \lfloor \log n \rfloor$*, where* n *is the number of requests.*

Proof. Without loss of generality n is a power of 2. The set of servers is defined as follows: There is one server at 0, one server at n and 2 servers at each of the even integers in $(0, n)$. The requests come in stages. In the first stage all odd points in $(0, n)$ are requested. Consider the request at $2j + 1$. Using the Locality Property, we see that the γWFA can service it by either a server on the left, at $2j$, or a server at the right, at $2j + 2$, because they are at equal distance from the request. We want to allow the adversary to break the tie. This can be enforced by perturbing the request slightly. In particular, we service all requests with the servers at positions $2 + 4k$, as shown in Figure 2. Notice now that the remaining servers are in a configuration similar to the initial one, only now they

Fig. 2. Stage 1 (x denotes request and o denotes server)

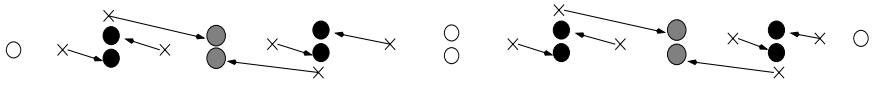


Fig. 3. Stage 2

are placed apart at distance 4 instead of 2. So we can repeat the construction. In particular, in the second stage all points $2 + 4k$ are requested and are serviced by the servers at points $4 + 8k$. We repeat it until only the server at 0 remains (in the last stage, a request is placed at $n/2$ which is serviced by the server at n). At the end, just to finish the construction we place a request at n and it is serviced by the only remaining server at 0.

The online cost at each stage is $n/2$ and the cost of the last request is n. There are $\log n$ stages, so the total online cost is $n + \frac{n}{2} \log n$. The total offline cost is easily seen to be $n/2$. The competitive ratio is $2 + \log n$. □

4 Upper Bound

We define the set of points crossed exactly j times by the optimal matching $M(A, R)$ as

$$B^j(A, R) = \{x : \beta_x(A, R) = j\}.$$

We also use the notation $B^-(A, R) = \cup_{j<0} B^j(A, R)$ and $B^+(A, R) = \cup_{j>0} B^j(A, R)$. To keep the expressions simple, we use the same symbols for the measure of these sets. Given these definitions, observe that

$$M(A, R) = \sum_j |j| \cdot B^j(A, R). \tag{4}$$

Thus for any balanced interval (s_1, s_2) which includes the set of servers A and set of requests R, we have $M(A+s_1, R+s_2) = M(A, R)+B^+(A, R)+B^0(A, R)-B^-(A, R)$ and $d(s_1, s_2) = B^+(A, R) + B^0(A, R) + B^-(A, R)$. Substituting these values in Theorem 1, we have:

$$2\gamma\, B^-(A, R) \le (\gamma + 1)\, d(s1, s2), \text{ and}$$
$$2\gamma\, B^+(A, R) \le (\gamma + 1)\, d(s1, s2).$$

Now we are ready to prove a generalization that will be very useful later.

Lemma 2. *Let (s_1, s_2) be a γWFA balanced interval which includes the set of servers A and set of requests R. For any $x \in (s_1, s_2)$*

$$2\gamma\, \|B^-(A, R) \cap [s_1, x]\| \le (\gamma + 1)\, d(s_1, x), \text{ and}$$
$$2\gamma\, \|B^+(A, R) \cap [x, s_2]\| \le (\gamma + 1)\, d(x, s_2).$$

Proof. The intuition behind the lemma is that since s_1 is an unmatched server, it was rejected in favor of other servers in the interval (s_1, s_2). This in turn indicates that $\|B^-(A, R) \cap [s_1, x]\|$ cannot be very large.

We prove only the first part, the second is similar. Suppose that the statement was true before the last request, $r_t \in (s_1, s_2)$, was matched to a server $s_r \in (s_1, s_2)$. Let $A_{t-1} = A - s_r$ and $R_{t-1} = R - r_t$. We consider two cases depending on whether s_r is to the left of r_t or not. The first case when $s_r < r_t$ is trivial because such a request increases the values $\beta_x(A_{t-1}, R_{t-1})$.

The other case, $r_t < s_r$, is more involved. We first notice that when x is in (s_1, r_t) then the lemma holds directly by induction. Also if the lemma holds for $x = s_r$ then it immediately holds for any $x \in [s_r , s_2)$ from the induction hypothesis for the balanced interval (s_r , s_2).

Therefore we concentrate on the case $x \in [r_t, s_r]$. To show that the lemma holds for $x = s_r$ we don't need the induction hypothesis, only the fact that γWFA services r_t with s_r instead of s_1:

$$\gamma M(A_{t-1} + s_1, R_{t-1} + r_t) + d(s_1, r_t) \ge \gamma M(A_{t-1} + s_r , R_{t-1} + r_t) + d(r_t, s_r).$$

With some work we can rewrite it as

$$2\gamma \, \|B^-(A,R) \cap [s_1, s_r\,]\| \leq (\gamma+1)\, d(s_1, s_r\,) - 2\, d(r_t, s_r\,), \tag{5}$$

which is slightly stronger than the lemma when $x = s_r$.

Finally, for the case $x \in [r_t, s_r\,)$, we use (5) and the second inequality of the inductive hypothesis for the interval $(x, s_r\,)$. Specifically, by induction on this interval, we get $2\gamma \, \|B^+(A_{t-1}, R_{t-1}) \cap [x, s_r\,]\| \leq (\gamma+1)\, d(x, s_r\,)$, which can be rewritten as

$$-2\gamma \, \|B^-(A,R) \cap [x, s_r\,]\| \leq -(\gamma-1) d(x, s_r\,) \tag{6}$$

Summing (5) and (6) we get

$$2\gamma \, \|B^-(A,R) \cap [s_1, x]\| \leq (\gamma+1) d(s_1, x) - 2d(r_t, x)$$

and the lemma follows. □

4.1 Optimal vs Pseudo-Optimal Matching

Suppose that after n requests, γWFA has matched R_n to servers S_n. The (offline) optimal matching though may match R_n to some other set S'_n. We call the optimal matching $M(S_n, R_n)$ the pseudo-optimal matching and denote its weight by $popt_n$ to distinguish it from the optimal matching $M(S'_n, R_n)$ whose weight is opt_n. Although γWFA cannot use the optimal set of servers (in general $S_n \neq S'_n$), it has the nice property that it uses an almost optimal set of servers.

Theorem 4. *For any set of servers and requests,*

$$opt_n \leq popt_n \leq \frac{\gamma+1}{\gamma-1} opt_n.$$

We now proceed to prove this theorem with the help of Lemma 2. To prove Theorem 4, we assume without loss of generality that all requests are in a balanced interval (s_1, s_2), otherwise we sum up the parts for each balanced interval. We need to compare $popt_n = M(S_n, R_n)$ and $opt = M(S'_n, R_n)$, for any S'_n such that $|S'_n| = |S_n| = |R|$. We can obtain the matching $M(S'_n, R_n)$ by putting requests at server points in $S_n - S'_n$ and matching them to servers in $S'_n \setminus S_n$: $M(S'_n, R_n) = M(S'_n \cup S_n, R_n + S_n - S'_n)$. Observe now that in the worst case the servers in S'_n that are outside the interval (s_1, s_2) are at s_1 or s_2. Thus, instead of proving Theorem 4, we shall prove the following stronger lemma.

Lemma 3. *Let (s_1, s_2) be a γ*WFA *balanced interval that contains the set of servers A which is matched to requests R. For any points $v_1, \ldots, v_{k+m}$ in (s_1, s_2)*

$$M(A + s_1^k + s_2^m, R + v_1 + v_2 + \cdots + v_{k+m}) \geq \frac{\gamma-1}{\gamma+1} M(A, R)$$

Proof. Assume $v_1 \le \cdots \le v_{k+m}$ and denote $v_0 = s_1$. We concentrate in showing the special case with $m = 0$ and then observe that the same proof applies to the general case. Define $\zeta_i^j = \|\{x : \beta_x(A,R) = j\} \cap [v_{i-1}, v_i]\|$ to be the measure of points between v_{i-1} and v_i which the optimal matching crosses j times (with appropriate sign). Applying Lemma 2 to interval (s_1, v_t) we obtain $\frac{\gamma-1}{\gamma+1}\sum_{i=1}^{t}\sum_{j<0}\zeta_i^j \le \sum_{i=1}^{t}\sum_{j\ge 0}\zeta_i^j$ and summing for $t = 1, \ldots, k$, we get:

$$\frac{\gamma-1}{\gamma+1}\sum_{t=1}^{k}\sum_{i=1}^{t}\sum_{j<0}\zeta_i^j \le \sum_{t=1}^{k}\sum_{i=1}^{t}\sum_{j\ge 0}\zeta_i^j,$$

which can be rewritten as

$$\frac{\gamma-1}{\gamma+1}\sum_{i=1}^{k}\sum_{j<0}(k+1-i)\zeta_i^j \le \sum_{i=1}^{k}\sum_{j\ge 0}(k+1-i)\zeta_i^j.$$

Using this and the assumption $\gamma > 1$, we can bound

$$\begin{aligned} &M(A,R) - M(A + s_1^k, R + v_1 + \cdots, v_k) \\ &= \sum_{i=1}^{k}\sum_{j}|j|\zeta_i^j - \sum_{i=1}^{k}\sum_{j}|j+k+1-i|\zeta_i^j \\ &\le \frac{2}{\gamma+1}\sum_{i=1}^{k}\sum_{j<0}|j|\zeta_i^j \end{aligned}$$

The last quantity is at most equal to $\frac{2}{\gamma+1}M(A,R)$ and the lemma follows (for $m = 0$). When $m > 0$ the proof is very similar but now the right-hand side includes also the terms with $j > 0$; this is still bounded by $\frac{2}{\gamma+1}M(A,R)$ and the lemma holds. □

This theorem implies almost immediately that γWFA is $O(n)$-competitive as follows: It is not hard using the β_x values to verify that $d(r_n, s_r\ ) \le M(S_{n-1}, R_{n-1}) + M(S_n, R_n)$ which in turn implies that the cost to service request r_n is bounded above by $popt_{n-1} + popt_n \le \frac{\gamma+1}{\gamma-1}(opt_{n-1} + opt_n)$. Since opt_n is non-decreasing function (which by the way is not true for $popt_n$) the claim follows. We however show a stronger result and in the process we will discover some important properties that may be useful to improve the competitive ratio. We will show that the competitive ratio is bounded (up to a constant factor) by the maximum number of crossing lines in the γWFA matching.

4.2 Alive vs Pseudo-Optimal

In this section, we take a closer look at the structure of the pseudo-optimal matching, and use this analysis to derive an $O(n)$ bound on the competitive ratio of γWFA.

We say that a point x contributes to $popt_n$ at time t if

$$0 \leq \beta_x(S_{t-1}, R_{t-1}) < \beta_x(S_t, R_t) \leq \beta_x(S_{t+1}, R_{t+1}), \ldots, \beta_x(S_n, R_n)$$

or

$$0 \geq \beta_x(S_{t-1}, R_{t-1}) > \beta_x(S_t, R_t) \geq \beta_x(S_{t+1}, R_{t+1}), \ldots, \beta_x(S_n, R_n)$$

We say that a point x contributes to $alive_n$ at time t if the same inequalities hold but not necessarily the leftmost ones (involving 0). For example, in Figure 4, x contributes to $alive_8$ at times 3,6,7, and 8 (which correspond to the bold lines; intuitively these are the lines that are visible from the point $(\infty, 0)$. In the same figure, x contributes to $popt_8$ at times 7 and 8 (which correspond to the bold lines after the last visit to the horizontal axis).

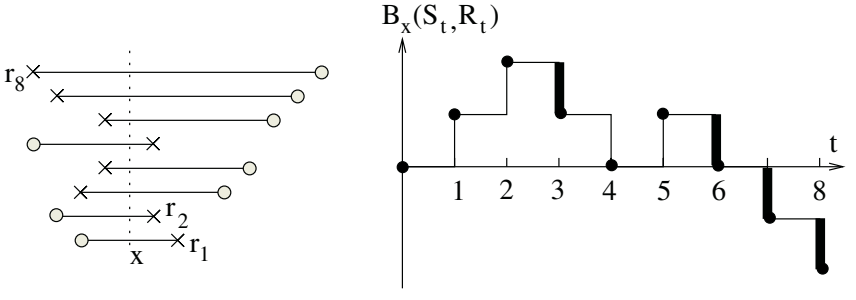


Fig. 4. $alive_n$ and $popt_n$

We say that a point contributes to $alive_n$ (or opt_n) k times when it contributes to $alive_n$ (or opt_n) at exactly k distinct times. For example, the point x above contributes 4 times to $alive_8$ and 2 times to $popt_8$. Clearly $popt_n$ is the measure of all points, each taken with multiplicity equal to the number of times that it contributes to $popt_n$. Similarly, $alive_n$ is the measure of all points, each taken with multiplicity equal to the number of times that it contributes to $alive_n$.

Theorem 5.

$$\frac{\gamma - 1}{2\gamma} alive_n \leq popt_n \leq alive_n.$$

Proof. The second inequality is obvious from the definitions. The first inequality is based on Lemma 2. Consider a request r_t which is serviced by request s_r and assume without loss of generality that $s_r < r_t$. The points that contribute to $alive_n$ at time t form an interval (s_r, q) for some point q. By Lemma 2 at least a fraction $\frac{\gamma-1}{2\gamma}$ of this interval has $\beta_x(A_{t-1}, R_{t-1}) \geq 0$. This fraction contributes also to $popt_n$ at time t. □

4.3 Alive vs Cost

We can now put everything together to obtain the desired upper bound. We have considered the four quantities: opt_n, $popt_n$, $alive_n$ and $cost_n$ (the last one is the weight of the online matching). Theorems 4 and 5 establish that the first three quantities are almost equal within some constant factors:

$$opt_n \le popt_n \le alive_n \le \frac{2\gamma}{\gamma-1}\frac{\gamma+1}{\gamma-1} opt_n$$

It is remarkable though that opt_n and $alive_n$ are non-decreasing while $popt_n$ is not. Furthermore, the cost to service request r_t is at most equal to $alive_t$ (each point in the interval (r_t, s_r) contributes to $alive_t$). Therefore we can bound the total $cost_n$ by

$$cost_n \le \sum_{t=1}^{n} alive_t \le n \cdot alive_n \le \frac{2\gamma}{\gamma-1}\frac{\gamma+1}{\gamma-1} \cdot n \cdot opt_n.$$

We showed

Theorem 6. *The* γWFA *has competitive ratio at most* $\frac{2\gamma}{\gamma-1}\frac{\gamma+1}{\gamma-1} \cdot n = O(n)$, *where* n *is the number of requests. In particular, when each point* x *is crossed by at most* k *lines of the matching of* γWFA, *the competitive ratio is* $O(k)$.

5 Conclusions and Open Problems

For the online matching problem on a line, we showed that the WFA has competitive ratio between $\Omega(\log n)$ and $O(n)$. We believe that the lower bound is tight and that the tools we developed here can be very useful. We don't know whether there exists some other algorithm which has constant competitive ratio.

One important property is the relation between *popt* and *opt* which states that although WFA may not use the optimal subset of servers —the one that minimizes the matching with the requests— it uses an almost optimal one (up to a constant factor). This may be true for arbitrary metric spaces.

For the Euclidean space R^d, the lower bound for the cow-path problem, $\Omega(n^{1-1/d})$, can be extended to the matching problem. Is this tight and what is the competitive ratio of WFA? Finally, randomized algorithms may have much better competitive ratio; the best randomized lower bound for arbitrary metric spaces is only $\Omega(\log n)$ [6].

References

1. Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Raw lins. Searching in the plane. *Information and Computation*, 106(2):234–252, October 1993.
2. J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards B*, 69B:125–130, 1965.

3. J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.
4. Bernhard Fuchs, Winfried Hochstättler, and Walter Kern. Online matching on a line. In Hajo Broersma, Ulrich Faigle, Johann Hurink, Stefan Pickl, and Gerhard Woeginger, editors, *Electronic Notes in Discrete Mathematics*, volume 13. Elsevier, 2003.
5. Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, May 1993. A Preliminary version appeared in the proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms, 1991, pages 234–240.
6. Bala Kalyanasundaram and Kirk Pruhs. Online network optimization problems. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 268–280. Springer-Verlag, 1998.
7. Bala Kalyanasundaram and Kirk Pruhs. Online transportation problem. *SIAM Journal of Discrete Mathematics*, 13(3):370–383, 2000. A Preliminary version appeared in Proceedings of the European Symposium on Algorithms, pages 484–493, 1995.
8. Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd ACM Annual Symposium on Theory of computing*, pages 352–358, 1990.
9. Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127:255–267, 1994. A preliminary version appeared as Technical Report 90–1143, Department of Computer Science, Cornell University, 1990.
10. L. Lovász and M. D. Plummer. *Matching Theory.* Elsevier Science Pub. Co., 1986.

How to Whack Moles*

Sven O. Krumke[1,**], Nicole Megow[2], and Tjark Vredeveld[1]

[1] Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization, Takustr. 7, D-14195 Berlin-Dahlem, Germany
{krumke,vredeveld}@zib.de
[2] Technische Universität Berlin, Strasse des 17. Juni 136, 10623 Berlin, Germany
nmegow@math.tu-berlin.de

Abstract. In the classical *whack-a-mole game* moles that pop up at certain locations must be whacked by means of a hammer before they go under ground again. The goal is to maximize the number of moles caught. This problem can be formulated as an online optimization problem: Requests (moles) appear over time at points in a metric space and must be served (whacked) by a server (hammer) before their deadlines (i.e., before they disappear). An online algorithm learns each request only at its release time and must base its decisions on incomplete information. We study the online whack-a-mole problem (WHAM) on the real line and on the uniform metric space. While on the line no deterministic algorithm can achieve a constant competitive ratio, we provide competitive algorithms for the uniform metric space. Our online investigations are complemented by complexity results for the offline problem.

1 Introduction

In the popular *whack-a-mole game* moles pop up at certain holes from under ground and, after some time, disappear again. The player is equipped with a hammer and her goal is to hit as many moles as possible while they are above the ground. Clearly, from the viewpoint of the player this game is an online optimization problem since the times and positions where moles will peek out are not known in advance. She has to decide without knowledge of the future which of the currently visible moles to whack and how to move the hammer into a "promising" position. What is a good strategy for whacking moles (if there exists any)? How much better could one perform if one had magical powers and knew in advance where moles will show up? How hard is it to compute an optimal solution offline? In this paper we investigate all of the above questions.

The *whack-a-mole problem* with popup duration $T \geq 0$ (WHAM_T) can be formulated in mathematical terms as follows: We are given a metric space $M = (X, d)$ with a distinguished origin $0 \in X$ and a sequence $\sigma = (r_1, \ldots, r_m)$ of requests. A server (hammer) moves on the metric space M at unit speed. It starts in the origin at time 0. Each request $r_j = (t_j, p_j, n_j)$ specifies a *release*

* Supported by the DFG Research Center "Mathematics for key technologies" (FZT 86) in Berlin.
** Research supported by the German Science Foundation (DFG, grant Gr 883/10).

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 192–205, 2004.
© Springer-Verlag Berlin Heidelberg 2004

time t_j and a point (hole) $p_j \in X$ where n_j moles pop up. A request r_j is served if the server reaches the point p_j in the time interval $[t_j, t_j + T]$. We will refer to $t_j + T$ as the *deadline* of the request. The goal is to whack as many moles as possible. If no confusion can occur we write only WHAM instead of WHAM$_T$.

An online algorithm learns about the existence of a request only at its release time. We evaluate the quality of online algorithms by *competitive analysis* [5], which has become a standard yardstick to measure the performance. An algorithm ALG for WHAM is called *c-competitive* if for any instance the number of moles whacked by ALG is at least $1/c$ times the number of moles caught by an optimal offline algorithm OPT. The first two of the questions raised above amount to asking which competitive ratios are achievable by online algorithms.

In this paper we mainly study the WHAM on two different metric spaces: the line and the uniform metric space. The line is the classical setup for the *whack-a-mole game*. The motivation for studying the uniform metric space (a complete graph with unit edge weights) is that "in practice" it does barely matter between which points the hammer is moved: the main issue is whether the hammer is moved (and to which point) or whether it remains at its current location.

Related work. The WHAM falls into the class of *online dial-a-ride problems*. In an online dial-a-ride problem objects must be transported between points in a metric space by a server of limited capacity. Transportation requests arrive online, specifying the objects to be transported and the corresponding source and destination. If for each request its source and destination coincide, the resulting problem is usually referred to as the online traveling salesman problem (OLTSP). The WHAM is the OLTSP with the objective to maximize the (weighted) number of requests served before their deadlines.

The OLTSP has been studied for the objectives of minimizing the makespan [1, 2, 7], the weighted sum of completion times [7, 13], and the maximum/average flow time [9, 14]. Since dial-a-ride problems (where sources and destinations need not coincide) can be viewed as generalizations of scheduling problems (see e.g. [1]), lower bounds for scheduling problems carry over. In [3], Baruah et al. show that no deterministic algorithm can achieve a constant competitive ratio for the scheduling problem of maximizing the number of completed jobs. Kalyanasundaram and Pruhs [11] show that for every instance at least one of two deterministic algorithms is constant competitive, and thus they provide a randomized algorithm which is constant competitive. However, it is not clear whether and how their results carry over to the more general class of dial-a-ride problems.

The WHAM has also been investigated in [10] under the name "dynamic traveling repair problem". The authors give two deterministic algorithms for the WHAM$_T$ in general metric spaces with competitive ratios that are formulated in terms of the diameter of the metric space. Their ratios translated into the notation used in this paper and restricted to the uniform metric space are $\frac{3T}{T-2}$ and $4\left\lceil \frac{2T}{T-1} \right\rceil \left(\left\lceil \frac{2T}{T-1} \right\rceil + 1 \right)$, respectively. We improve these results in several ways for the case of the uniform metric space. For instance, for popup duration

Table 1. Competitiveness results for the WHAM$_T$ on the uniform metric space where each mole stays T units of time above ground. Here, N denotes the maximum number of moles (possibly stemming from different requests) that are peeking out of the same hole, simultaneously, for a positive amount of time.

Popup duration	upper bound	lower bound
$T \geq 2$	$\frac{\lceil \lfloor T/2 \rfloor + T \rceil}{\lfloor T/2 \rfloor} \in [3,5]$ (see Figure 2)	2
$1 < T < 2$	$2N$	2
$T = 1$	2 (for all t_j integral) $2N$ (for general t_j)	2 (for all t_j integral) $2N$ (for general t_j)

$T = 2$, our algorithm IWTM achieves a competitive ratio of 3, while the results in [10] yield a ratio of 80. Moreover, for $T = 1$ our algorithms are the first competitive ones, since the bounds of [10] cannot be applied. The paper [10] also shows that there is a metric space in which no deterministic algorithm can achieve a constant competitive ratio. We show that this results is already true on the real line.

Our contribution. The contributions of this paper are twofold. First, we provide complexity results for the offline problem OFFLINE-WHAM. We derive a dynamic program for the OFFLINE-WHAM on unweighted graphs with integral release times and deadlines, which runs in time $\mathcal{O}(nm(T+m)(\Delta+1)^{2T})$, where n is the number of nodes in the space, m denotes the number moles and Δ is the maximum degree. The algorithm runs in polynomial time, if $(\Delta+1)^{2T}$ is bounded by a polynomial in the input size. We complement our solvability result by NP-hardness results for some special cases of the OFFLINE-WHAM.

Our main contribution lies in the analysis of the online problem WHAM. We show that no deterministic algorithm for the WHAM on the line can achieve a constant competitive ratio. From the viewpoint of the *whack-a-mole* player, the situation is much better on the uniform metric space. Our results for this case are summarized in Table 1. In particular, we provide the first competitive algorithm for short popup durations $T = 1$ and substantially decrease the known competitive ratio for $T \geq 1$. Our competitiveness results are complemented by lower bounds on the competitive ratio of arbitrary deterministic algorithms. The upper bounds hold against the most powerful adversary, whereas the lower bounds are shown against the *non-abusive adversary* [14], which is the most restricted adversary that we consider.

2 The Complexity of Offline Whack-a-Mole

In this section we investigate the complexity of the offline problem OFFLINE-WHAM where all moles and their respective release dates are known in advance.

We first give a polynomial-time algorithm for a special class of the problem. Then, we show that OFFLINE-WHAM is NP-hard on the line and on the star graph. In this section we slightly diverge from the notation used for the online problem in allowing more general deadlines $d_j \geq t_j$ for the requests than just $d_j = t_j + T$, where T is the popup duration. In this more general context T will denote the maximum popup duration of any mole.

2.1 When Whacking Is Easy

We consider the following scenario: The metric space $M = (X, d)$ has n points and is induced by an undirected unweighted graph $G = (V, E)$ with $V = X$, i.e., for each pair of points from the metric space M we have that $d(x, y)$ equals the shortest path length in G between vertices x and y. We also assume that for each mole the release date $t_j \geq 1$ and the deadline d_j are integers.

Theorem 1. *Suppose that the metric space is induced by an unweighted graph of maximum degree Δ. Then, the* OFFLINE-WHAM *with integral t_j and d_j can be solved in time $\mathcal{O}(nm(T+m)(\Delta+1)^{2T})$, where $T := \max_{1 \leq j \leq m}(d_j - t_j)$ is the longest time a mole stays above the ground; n denotes the number of vertices in the graph and m is the number of requests.*

Proof. The time bound claimed is achieved by a simple dynamic programming algorithm. Let $0 < t_1 < t_2 < \cdots < t_k$ with $k \leq m$ be the (distinct) times where moles show up. We set $t_0 := 0$.

The idea for a dynamic programming algorithm is the following: For each relevant point t in time and each vertex $v \in V$ we compute the maximum number of moles caught subject to the constraint that at time t we end up at v. Essentially the only issue in the design of the algorithm is how one keeps track of moles that have been whacked "on the way". The key observation is that for any time t that we consider the only moles that need to be accounted for carefully are those ones that have popped up in the time interval $[t-T, t]$. Any mole that popped up before time $t-T$ will have disappeared at time t anyway. This allows us to use a limited memory of the past.

Given a vertex v, a *history track* is a sequence $s = (v_1, v_2, \ldots, v_k = v)$ of vertices in G such that for $i = 1, \ldots, k$ we have $d(v_i, v_{i+1}) = 1$ whenever $v_i \neq v_{i+1}$. We define the time-span of the history track s to be $\bar{d}(s) = k$. The history track s encodes a route of starting at vertex v_1 at some time t, walking along edges of the graph and ending up at v at time $t + \bar{d}(s)$ with the interpretation if $v_i = v_{i+1}$ we remain at vertex v_i for a unit of time. Notice that in an unweighted graph with maximum degree at most Δ, there are at most $(\Delta+1)^L$ history tracks of length $L \in \mathbb{N}$ ending at a specific vertex v.

Given the concept of a history track, the dynamic programming algorithm is straightforward. For $t \in \{t_0, \ldots, t_k\}$, $v \in V$ and all history tracks s, with $\bar{d}(s) = \min(t, T)$, ending in v at time t, we define $M[t, v, s]$ to be the maximum number of moles hit in any solution that starts in the origin at time 0, ends in v at time t, and follows the history track s for the last $\bar{d}(s)$ units of time.

The values $M[0, v, s]$ are all zero, since no mole raises its head before time 1. Given all the values $M[t, v, s]$ for all $t = t_0, \ldots, t_{j-1}$, we can compute each value $M[t_j, v, s]$ easily.

Assume that $t_j \leq t_{j-1} + T$. Then, from the history track s we can determine a vertex v' such that v' must have been at vertex v' a time t_{j-1}. This task can be achieved in time $\mathcal{O}(T)$ by backtracking s. The value $M[t_j, v, s]$ can now be computed from the $\mathcal{O}((\Delta+1)^T)$ values $M[t_{j-1}, v', s']$ by adding the number of moles whacked and subtracting the number of moles accounted for twice. The latter task is easy to achieve in time $\mathcal{O}(T+m)$ given the history tracks s and s'. Hence, the time needed to compute $M[t_j, v, s]$ is $\mathcal{O}((T+m)(\Delta+1)^T)$.

It remains to treat the case that $t_j > t_{j-1} + T$. Let $t := t_{j-1} + T$. Notice that no mole can be reached after time t and before time t_j, since all moles released no later then t_{j-1} will have disappeared by time t. Any solution that ends up at vertex v at time t_j must have been at some vertex v' at time t. We first compute the "auxiliary values" $M[t, v', s']$ for all $v' \in V$ and all history tracks s by the method outlined in the previous paragraph. Then, the value $M[t_j, v, s]$ can be derived as the maximum over all values $M[t, v', s']$, where the maximum ranges over all vertices v' such that v can be reached by time t_j given that we are at v' at time t and given the histories s and s' (which must coincide in the relevant part).

Since the dynamic programming table has $\mathcal{O}(nm(\Delta+1)^T)$ entries, the total time complexity of the algorithms is in $\mathcal{O}(nm(T+m)(\Delta+1)^{2T})$. □

The above dynamic program can easily be adjusted for metric spaces induced by weighted graphs with integral edge weights. Each edge e is then replaced by a path of $w(e)$ vertices, where $w(e)$ denotes the length of edge e. The time bound for the above procedure becomes then $\mathcal{O}(\bar{n}m(T+m)(\Delta+1)^{2T})$, where $\bar{n} = n + \sum_{e \in E}(w(e) - 1)$. Hence, whenever $(\Delta+1)^{2T}$ is pseudo-polynomially bounded, OFFLINE-WHAM can be solved in pseudo-polynomial time on these weighted graphs.

2.2 When Whacking Is Hard

It follows from Theorem 1 that OFFLINE-WHAM can be solved in polynomial time if $(\Delta+1)^{2T}$ is bounded by a polynomial in the input size. On the other hand, the problem on a graph with unit edge weights, all release times zero and all deadlines equal to n, the number of holes, contains the Hamiltonian Path Problem as a special case. Thus, it is NP-hard to solve, see e.g. [15].

Another special case of the OFFLINE-WHAM is obtained when at most one mole is in a hole at a time, the metric space is the line and release dates as well as deadlines are general. Then this problem is also NP-hard by a reduction from PARTITION, as mentioned in [6].

The first case we consider is the weighted version of OFFLINE-WHAM on the line where multiple moles remain above ground for a fixed time.

Theorem 2. OFFLINE-WHAM *on the line is NP-hard even if all moles stay above ground is equal for all moles, i.e.,* $d_i - t_i = d_j - t_j$ *for all requests* r_i, r_j.

Proof. We show the theorem by a reduction from PARTITION, which is well known to be NP-complete to solve [12, 8]. An instance of PARTITION consists of n items $a_i \in \mathbb{Z}^+$, $i = 1, \ldots, n$, with $\sum_i a_i = 2B$. The question is whether there exists a subset $S \subset \{1, \ldots, n\}$, such that $\sum_{i \in S} a_i = B$.

Given an instance of PARTITION, we construct an instance I_{WHAM} for OFFLINE-WHAM, with $m = 3n$ requests. Let $B = \frac{1}{2}\sum_i a_i$ and $K = B + 1$. The time each mole stays above ground is $T = 2B$. There are $2m$ requests r_i^+ and r_i^-, $i = 1, \ldots, m$ where $r_i^{\pm}$ is released at time $(2i-1)K$ and has deadline $(2i-1)K+T$. The position of r_i^+ is $K+a_i$ with weight $K+a_i$, and the position of r_i^- equals $-K$ with weight K. Finally, there are m requests r_i^0 in the origin, where r_i^0 is released at time $2iK$, has deadline $2iK + T$, and weight K.

We claim that at least $2nK + B$ moles can be whacked if and only if I is a YES-instance for PARTITION.

Let S be a partition of I, i.e., $\sum_{i \in S} a_i = B$. Then whacking the moles of requests in the order $(r_1^{\alpha_1}, r_1^0, \ldots, r_n^{\alpha}, r_n^0)$, where $\alpha_i = +$ if $i \in S$ and $\alpha_i = -$ if $i \notin S$, is feasible and yields the desired bound, as tedious computation can show.

Suppose conversely that there exists a route for the whacker such that it reaches at least $2nK + B$ moles. Notice that as the locations of the holes of requests r_i^+ and r_i^- are at least $2K > 2B$ apart, the whacker can whack at most one of these requests. The moles of requests r_i^+ and r_i^- pop up after time $t_{i-1}^+ + T$, and therefore the whacker cannot catch the moles of request r_{i-1}^+ and r_i^+ at the same time. The same is true for requests r_{i-1}^0 and r_i^0. Suppose the whacker moves to the hole of r_i^+ or r_i^- after first whacking the moles of r_i^0. The earliest possible arrival time in the mole is at least $2iK + K = (2i+1)K$ and by this time the moles of r_i^+ and r_i^- have gone down again. Hence, when whacking r_i^0 and either r_i^+ or r_i^-, the request r_i^+ or r_i^- need to be whacked before r_i^0. Not whacking the moles of r_i^0 or none of r_i^+ and r_i^-, results in a tour in which at most $(2n-1)K + 2B < 2nK + B$ can be caught. Therefore, the whacker needs to reach all moles popping up in the origin and for each i it also needs to whack all moles of either r_i^+ or r_i^-. Hence, by the above considerations we know that when at least $2nK + B$ moles are whacked, the whacker needs to hit first the moles of r_i^+ or r_i^- and then those of r_i^0 before going to the hole of request r_{i+1}^+ or r_{i+1}^-.

Let $S = \{i : \text{moles of } r_i^+ \text{ are whacked}\}$ be the set of requests served in the positive part of the line. We claim that $\sum_{i \in S} a_i = B$. Obviously $\sum_{i \in S} a_i \geq B$ since the number of moles whacked is at least $2nK + B$. Suppose that $\sum_{i \in S} a_i > B$ and let $S' \subseteq S$ be the smallest subset of S such that if $i, j \in S$ with $i < j$ and $j \in S'$ then $i \in S'$ and $\sum_{i \in S'} a_i > B$ and let $k = \max S'$. Then $\sum_{i \in S' \setminus \{k\}} a_i \leq B$. The whacker leaves the origin for request r_k^+ at time $2(k-1)K + 2\sum_{i \in S' \setminus \{k\}} a_i \leq 2(k-1)K + 2B < t_k^0$. The next time the whacker reaches the origin is $2kK + \sum_{i \in S'} a_i > 2kK + 2B$ and by then the moles of request r_k^0 have gone under ground. Hence, it cannot reach the moles of request r_k^0 and is not able to whack $2nK + B$ moles. □

Our next hardness result concerns the case of a star graph.

Theorem 3. OFFLINE-WHAM *is NP-hard on a star graph with arbitrary edge lengths. This result holds, even if at any moment in time at most one mole is peeking out of hole.*

Proof. Let $(a_1, \ldots, a_n)$ be an instance for PARTITION and let $B = \frac{1}{2}\sum_i a_i$. We construct a star graph with $n+1$ leaves. The length of the center to leaf i is equal to a_i, for $i = 1, \ldots, n$ and the length of the $(n+1)$st leg is equal to 1. The request sequence is as follows. At time $t = 0$, a single mole is released in each of the leaves $j = 1, \ldots, n$. The popup duration of each mole is equal to $4B + 2$. At time $2B + 1$ a mole appears in leaf $n + 1$ with zero popup duration. Finally, at time $4B + 2$ a mole pops up in the center, also with zero popup duration.

If we have a YES-instance of PARTITION, i.e., there exists a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = B$, the whacker can catch all moles on the following route. First she whacks the moles in the leaves $i \in S$ in an arbitrary order and arrives at the center at time $2B$. Then, she whacks the mole in leaf $n+1$ after which all the remaining leaves can be visited and the whacker returns in the center at time $2 + \sum_i a_i$, at which she can whack the mole that pops up there.

If there exists a route in which the whacker can reach all moles, then by time $2B + 1$ she needs to be in leaf $n + 1$. Therefore, no later than time $2B$ she needs to be in the center. Let S be the set of moles whacked before this time. After whacking the mole in leaf $n + 1$ and returning to the origin, there is still $2\sum_i a_i - 2B$ time left to whack all the unwhacked moles and return in the center. Therefore, it must hold that $\sum_{i \in S} a_i = B$. □

3 Whack-a-Mole on the Line

In this and the following section we investigate the existence of competitive algorithms for the WHAM. Our results are not only established for the standard adversary, the optimal offline algorithm, but also for the more restricted adversaries of [4, 14], which we recall briefly.

The optimal offline algorithm is often considered as an adversary, that specifies the request sequence in a way that the online algorithm performs badly. Besides the ordinary adversary that has unlimited power, there exist several adversaries in the literature that are restricted in their power. The *fair adversary*, introduced by [4], may move at any time only within the convex hull of all requests released so far. An even more restricted adversary is the *non-abusive adversary* of [14]. This adversary is defined on the line, and it may only move into a certain direction if there is still a pending request in this direction. For WHAM we extend this definition by restriction that the adversary may only move in the direction of a request that it can reach before the deadline of this request. A natural extension to other metrics is an adversary that may only move on a direct path to a pending request, which deadline can be met.

Theorem 4. *Let $T \geq 0$ be arbitrary. No deterministic online algorithm can achieve a constant competitive ratio for* WHAM$_T$ *on the line.*

Proof. Consider an online algorithm ALG and assume w.l.o.g. that its position at time T is $p_{\text{ALG}}(T) \leq 0$. The sequence consists of one mole, which pops up at time T at position $T+1$. As the adversary can be at position T by time T, it has killed the mole by time $T+1$. ALG, on the other hand, can not be in $T+1$ before time $2T+1$ and by then the mole has gone under ground again. □

In the proof above the adversary abuses its power in the sense that it moves to a point where a mole will pop up without revealing this information to the online whacker. Theorem 4 can be extended to both, the fair and the non-abusive adversary as is shown in the following theorem.

Theorem 5. *Let $T \geq 0$ be arbitrary. No deterministic online algorithm can achieve a constant competitive ratio for the* WHAM$_T$ *on the line against a non-abusive adversary.*

Proof. Consider an arbitrary deterministic online algorithm. At time 0 two moles appear: one in T and the other in $-T$, both going down at time T. If the online whacker does not reach any mole by time T then the sequence stops. Otherwise, we assume w.l.o.g. that the online whacker is in position $p_{\text{ALG}}(T) = -T$ at time T. ADV is at that time in position T. Then, from time $t = T$ onwards a request is given in $t+1$ at each integral time $T, T+1, T+2, \ldots$. The adversary can whack all these moles, whereas the online whacker is not able to meet any of the deadlines. □

4 Whack-a-Mole on the Uniform Metric Space

Recall that the uniform metric space is induced by a complete graph with unit edge weights. The number of vertices in this graph is n. Observe that for $T < 1$ trivially no deterministic algorithm can be competitive against the standard or fair adversary. In case of the non-abusive adversary, the situation is trivial again, since the adversary can never catch any mole except for those in the origin.

4.1 How Well We Can't Whack

We remark that a lower bound of 2 for the WHAM on the uniform metric space has been derived in [10]. Our construction uses fewer nodes in the metric space and, more important, there is no positive amount of time where more than one request is available at a single hole. Also, note that the lower bounds are shown against the most restricted adversary of those defined in the previous section.

Theorem 6. *Let $n \geq 3T+2$, that is, $T \leq (n-2)/3$. No deterministic online algorithm for the* WHAM$_T$ *can achieve a competitive ratio smaller than* 2 *against a non-abusive adversary.*

Proof. At each time $t = 0, \ldots, T-1$ the adversary releases two moles: one in $p_{t,1} = 2t+1$ and the other in $p_{t,2} = 2t+2$. At time $t = T, T+1, \ldots$ three moles

are released: two moles are released in empty holes $p_{t,1}$ and $p_{t,2}$ and the third mole, either, in $p_{t,3} = p_{t-T,2}$ if ALG is in $p_{t-T,1}$ at time t, or, in $p_{t,3} = p_{t-T,1}$, otherwise. Note that at time t, at most $3T$ moles have deadline at least t, and as $n \geq 3T + 2$, there are at least two holes left with no moles at time t.

ALG cannot whack more than one mole per time unit, whereas ADV can kill two moles at a time from time $t = T$ onwards. □

In the sequel the maximum number of moles available at a single hole will play a crucial role. We define N to be the maximum number of moles peeking out of the same hole for a positive amount of time. Notice that with this definition, there might be a moment in time, where in fact $2N$ moles are above ground at the same place. Consequently, an algorithm might whack up to $2N$ moles in a single step.

Theorem 7. *If at most N moles can be in the same hole for a positive amount of time, then any deterministic online algorithm for the* WHAM_1 *against a non-abusive adversary has a competitive ratio no less than* $2N$.

Proof. After an initial step, a non-abusive adversary constructs a sequence consisting of phases such that in each phase it whacks at least $2N$ times as many moles as ALG does. Each phase starts at a time t when the adversary arrives in a hole. We denote by t' the latest deadline of the moles that are in this hole at time t. Note that $t \leq t' < t + 1$, since the popup duration is 1. There are two possible positions for ALG to be at time t:

Case (a) ALG is in a vertex point different from the position of ADV;
Case (b) ALG is on an edge.

Moreover, if there are at the beginning of the phase some pending requests released before time t then ALG cannot reach any of them.

In Case (a), two moles are released at time t in holes where neither ALG nor ADV are. If ALG does not immediately go to one of these moles, it cannot whack any of them, whereas the adversary catches one of these moles. Otherwise, at time $\bar{t} = \max\{t', t + 1/2\}$ the adversary releases N moles in his current position and N moles in a hole v that is not incident to the edge on which ALG is. Thus, ALG cannot whack any of them. Hence, it whacks at most one mole, whereas ADV reaches $2N$ moles by remaining in his position until time $\bar{t}$ and then moving to v.

In Case (b), ALG is in the interior of an edge and thus, it cannot reach any vertex point which is not incident to this edge by time $t + 1$. The adversary releases one mole in a free hole, i.e., a vertex point where no mole is and which is not incident to the edge on which ALG is. Hence, ALG does not whack any mole, and ADV hits one mole.

An initial sequence consisting of two requests in two different holes each releasing a single mole ensures that we end up either in Case (a) or Case (b). This completes the proof. □

Note that in the proof of the above lower bound we use the fact that the release dates may be non-integral. As we see in the next section, this restriction is essential, because for integral release dates we are able to show that there exists a 2-competitive algorithm.

4.2 How Well We Can Whack

In this section we propose simple algorithms for WHAM and give performance guarantees for the online problem on a uniform metric space.

First Come First Kill (fcfk)
At any time t, move to a hole which contains a request with earliest release date, breaking ties in favor of the point where the most moles are above ground. If none of the moles that are above ground can be killed by the algorithm, then the whacker does not move.

Theorem 8. *Let $T \geq 1$. Consider the* WHAM$_T$ *with at most N moles peeking out of one hole for a positive amount of time.* FCFK *is $2N$-competitive, which is tight.*

Proof. Partition the input sequence into maximal subsequences, such that each subsequence consists of requests that FCFK serves continously, i.e., it is constantly moving between holes. We show that OPT whacks at most $2N$ times as many moles as FCFK does for each subsequence from which the theorem follows.

Consider such a subsequence σ'. We denote by C_j^{ALG} the time where algorithm ALG whacks request r_j. If r_j is not caught, then we set $C_j^{\mathrm{ALG}} = \infty$. Define

$$t_{\max} = \max\{\, C_j^{\mathrm{OPT}} : r_j \in \sigma' \text{ and } C_j^{\mathrm{OPT}} < \infty \,\}.$$

We define $t_{\min}$ such that $t_{\max} - t_{\min}$ is integral and $\min_{j\in\sigma'} t_j \leq t_{\min} < \min_{j\in\sigma'} t_j + 1$. In each interval $(t, t+1]$ for $t = t_{\min}, \ldots, t_{\max} - 1$, FCFK hits at least one mole and OPT cannot whack more than $2N$ moles. It remains to show that the moles which are reached by OPT before $t_{\min}$ can be compensated for by FCFK.

If FCFK whacks its last mole of σ' no later than time $t_{\max}$, then OPT catches at most N moles in the interval $(t_{\max} - 1, t_{\max}]$ since no new request can be released at $t_{\max}$ due to the maximality of the subsequence σ'. Moreover, OPT can kill at most N moles in the interval $(\min_{j\in\sigma'} t_j, t_{\min}]$. Therefore, the number of moles reached by OPT during the period before $t_{\min}$ can be accounted for by the moles caught in the last interval by OPT and thus, sum up to at most $2N$.

On the other hand, if FCFK still whacks a mole from σ' after time $t_{\max}$, the number of moles caught by OPT during the first period is at most N times the number of moles hit by FCFK after $t_{\max}$.

Lemma 1 implies that the competitive ratio of $2N$ is tight for FCFK. □

Lemma 1. *Let $T \geq 1$ be an integer. Consider the* WHAM$_T$ *with at most N moles peeking out of one hole for a positive amount of time.* FCFK *has no competitive ratio less than $2N$ against a non-abusive adversary.*

Proof. At time $t = 0$, the adversary releases T requests in holes $1, \ldots, T$, each of them with weight 1. At time $t = 1/2$, in hole $2T + 1$, a request is released with N moles. At time t, for $t = 1, \ldots, T - 1$, one request in hole $T + t$ is given with one mole and at time $t + 1/2$ a request with N moles is given in $2T + 1 + t$.

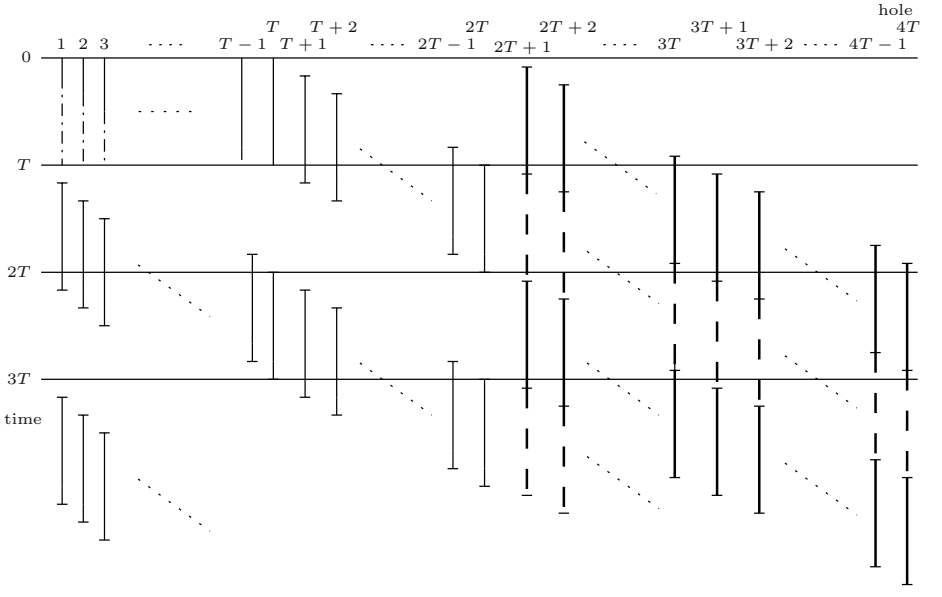


Fig. 1. Lower bound sequence for FCFK. Each request is represented by a vertical line between the release date and deadline. Thick lines illustrate requests with N moles, the thin lines depict requests with single moles. The line segment is dashed after the request has been served by ADV, and dash-dotted after being served by FCFK and ADV. Notice that from time T onwards, FCFK serves all its requests by their deadlines.

At time t, for $t = T, T+1, \ldots$, one mole is popping up in hole $1 + (T + t - 1) \bmod 2T$. And at time $t + 1/2$ two requests are given, each with N moles: one in $2T + 1 + (t \mod T)$ and one in $3T + 1 + (t \mod T)$. This sequence is visualized in Figure 1.

Up to time T, FCFK whacks the moles released at time 0. After time T it moves to the hole with the earliest released request that it can reach. As the requests with N moles are released $1/2$ time unit later than the requests with a single mole, FCFK is not able to whack any of the higher weighted requests. Hence, it catches in each unit length time interval one mole. In each unit length interval from time $T + 1/2$ onwards, there is one hole where a request with N moles has its deadline and a new request with N moles is released. Hence, ADV can whack $2N$ moles in every unit length time interval after time T. □

Recall that no deterministic online algorithm can be better than 2-competitive (Theorem 6). Hence, by Theorem 8 we know that FCFK is optimal in the unweighted setting, i.e., at most one mole can be peeking out of a single hole for a positive amount of time. Also, in the weighted case with $T = 1$, FCFK is optimal by Theorem 7. For the special case of integral release dates and $T = 1$, FCFK obtains a competitive ratio of 2 even in the weighted setting.

Theorem 9. *Consider the* WHAM$_1$ *where each mole stays* $T = 1$ *time unit above ground.* FCFK *is* 2*-competitive if all release dates are integral.*

Proof. Due to the integral release dates, both, OPT and FCFK are in holes at integral points in time. Moreover, OPT serves at most two requests released at the same time because of the unit popup duration. FCFK on the other hand, whacks at least the moles of one request released at a certain time and by definition it chooses the request with the highest number of moles. Therefore, it reaches at least half of the moles whacked by the optimal offline algorithm. □

Obviously, FCFK's flaw lies in ignoring all requests with a later deadline even though they could contribute with a higher weight to the objective value. In order to overcome this drawback we consider an other algorithm which we call *Ignore and Whack the Most* (IWTM). In this algorithm, we divide the time horizon in intervals of length $l = \lfloor \frac{T}{2} \rfloor$, and we denote these intervals by $I_i = ((i-1)l, il)$, for $i = 0, 1, 2, \ldots, L$, where I_L is the last interval in which moles are whacked. We say that at time t, the *current interval* is the interval I_i for which $t \in I_i$. Note that these intervals only have a positive length for $T \geq 2$.

When formulating the algorithm IWTM we allow the algorithm to whack only a subset of the moles available at a certain hole. Although our problem definition would force all moles at v to be whacked, this condition can be enforced within the algorithm by keeping a "virtual scenario".

Ignore and Whack the Most (iwtm)
At any time when the whacker is at a hole, it moves to the hole with the highest number of pending moles released before the current interval that can still be reached in time. Only those moles will be whacked at the hole.

Theorem 10. *Let* $T \geq 2$ *and* $c = \frac{\lceil \lfloor T/2 \rfloor + T \rceil}{\lfloor T/2 \rfloor}$. IWTM *is* c*-competitive for the* WHAM$_T$.

Proof. Let k_i denote the number of moles released in interval I_i, whacked by OPT, and let h_i denote the number of moles whacked by IWTM during interval I_i. Then

$$\text{OPT}(\sigma) = \sum_i k_i, \qquad \text{and} \qquad \text{IWTM}(\sigma) = \sum_i h_i. \tag{1}$$

Moreover, since no moles are released in the last interval I_L, it follows that $k_L = 0$.

First note that IWTM is at integral time points always in a hole. Therefore, during interval I_{i+1} it can visit l holes. If it visits less than l holes, then the number of requests released in interval I_i is less than l. Hence, OPT cannot kill more than h_{i+1} moles of those released in I_i.

Conversely, suppose that IWTM visits exactly l holes during I_{i+1}. The optimum can visit at most $\lceil l + T \rceil$ holes of requests released in interval I_i. By definition IWTM serves the l holes with the highest weight of pending requests

released at or before time il. Therefore, $h_{i+1} \geq (l/\lceil l+T \rceil)k_i$. Hence, by Equations (1), we know that

$$\text{IWTM}(\sigma) \geq (l/\lceil l+T \rceil)\text{OPT}(\sigma).$$

Recall that $l = \lfloor T/2 \rfloor$. For T ranging from 2 to 20, the values of the competitive ratio are depicted in Figure 2. □

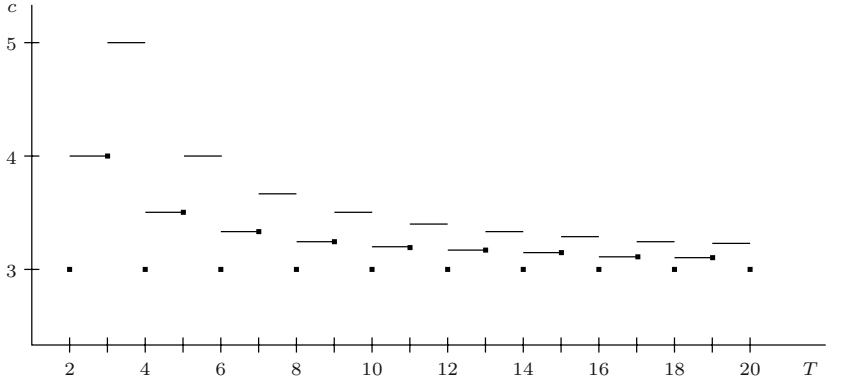


Fig. 2. Competitive ratio for IWTM for $T \in [2, 20]$.

5 Concluding Remarks

We have provided the first competitive algorithms for the WHAM on the uniform metric space for small popup duration and improved known competitiveness results for larger values. For the case of non uniform popup durations a natural extension of the FCFK algorithm would be the *Whack the Wimp* algorithm: always move to a hole containing a request with earliest deadline (breaking ties just as in FCFK). If all popup durations are at least 1, then using the same analysis as for FCFK we can show the same bounds on the uniform metric space. In the case that there are popup durations less than 1, no deterministic algorithm can be constant competitive.

Our lower bound results show that the situation on the real line is hopeless in terms of competitiveness, at least for deterministic algorithms. This raises the question whether randomized algorithms can do better.

References

1. N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650. Springer, 2000.
2. G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line traveling salesman. *Algorithmica*, 29(4):560–581, 2001.

3. S. Baruah, J. Haritsa, and N. Sharma. On-line scheduling to maximize task completions. *The Journal of Combinatorial Mathematics and Combinatorial Computing*, 39:65–78, 2001.
4. M. Blom, S. O. Krumke, W. E. de Paepe, and L. Stougie. The online-TSP against fair adversaries. *Informs Journal on Computing*, 13(2):138–148, 2001.
5. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
6. W. E. de Paepe, J. K. Lenstra, J. Sgall, R. A. Sitters, and L. Stougie. Computer-aided complexity classification of dial-a-ride problems. *Informs Journal on Computing*, 2003. To appear.
7. E. Feuerstein and L. Stougie. On-line single server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability (A guide to the theory of NP-completeness)*. W.H. Freeman and Company, New York, 1979.
9. D. Hauptmeier, S. O. Krumke, and J. Rambau. The online dial-a-ride problem under reasonable load. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2000.
10. S. Irani, X. Lu, and A. Regan. On-line algorithms for the dynamic traveling repair problem. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 517–524, 2002.
11. B. Kalyanasundaram and K. R. Pruhs. Maximizing job completions online. In *Proceedings of the 6th Annual European Symposium on Algorithms*, volume 1461 of *Lecture Notes in Computer Science*, pages 235–246. Springer, 1998.
12. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
13. S. O. Krumke, W. E. de Paepe, D. Poensgen, and L. Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295:279–294, 2003.
14. S. O. Krumke, L. Laura, M. Lipmann, A. Marchetti-Spaccamela, W. E. de Paepe, D. Poensgen, and L. Stougie. Non-abusiveness helps: An $O(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, volume 2462 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2002.
15. D. Poensgen. *Facets of Online Optimization*. Dissertation, Technische Universität Berlin, Berlin, 2003.

Online Deadline Scheduling: Team Adversary and Restart*

Jae-Ha Lee**

Computer Engineering Department, Konkuk University, Republic of Korea
jaehalee@konkuk.ac.kr

Abstract. We study the competitiveness of online deadline scheduling problems. It is assumed that jobs are non-preemptive and we want to maximize, in an online manner, the sum of the length of jobs completed before their deadlines. When there is a single machine, Goldwasser [4] showed that the optimal deterministic competitiveness of this problem is $2 + \frac{1}{k}$, where each job of length L can be delayed for at least $k \cdot L$ before it is started, while still meeting its deadline.
In this paper we generalize the framework of the above problem in two ways: First we replace the adversary by a set of m *weak adversaries*, each of which can generate arbitrary jobs that never overlap each other. Assuming that job sequence is generated by the team of m weak adversaries for some fixed m, we present a tight analysis of an optimal online algorithm by extending the previous analysis by Goldwasser. Next we allow online algorithms to abort the currently running job and to restart it from the scratch, if it can meet the deadline. This capability of abort and restart is different from preemptiveness because only jobs that are scheduled without interrupt are regarded to be successfully scheduled. We present a constant competitive algorithm for this model.

Keywords: Analysis of algorithms; Online algorithm; Real-time scheduling

1 Introduction

Online Admission Control. Imagine there is one resource (communication link, CPU, etc.) that services an online sequence of jobs. It is assumed that each job has a processing time and a *firm deadline* by which the job should be completed. A job is *accepted* if it runs *non-preemptively* (without interrupt) and is completed before its deadline. We aim to find a schedule of jobs that maximizes the sum of the lengths of accepted jobs. We consider the online problem.

Model. A job $\mathcal{J}_j$ is defined to be a triple of non-negative integers $\langle r_j, p_j, d_j \rangle$, where r_j is the arrival time, p_j is the length of the processing time, and d_j is

* This research was supported by grant No. R08-2003-000-10569-0(2003) from the Basic Research Program of the Korea Science and Engineering Foundation.
** Also supproted in part by University IT Research Center Project and by Faculty Research Fund of Konkuk University.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 206–213, 2004.
© Springer-Verlag Berlin Heidelberg 2004

the deadline by which the job should be completed. Let $e_j = d_j - p_j$ denote the *expiration time* – the last possible time at which $\mathcal{J}_j$ could be started while still meeting the deadline constraint – and let $s_j = d_j - p_j - r_j$ denote the *slack* (called *laxity* in [5]), which is the amount of time between r_j and the expiration time e_j. The gain of an algorithm A, denoted $Gain(A)$, is the sum of the length of the accepted jobs. The *competitive ratio* of an online algorithm A is the worst-case ratio between $Gain(A)$ and the gain of the optimal algorithm with full knowledge of future jobs. An online algorithm is *c-competitive* if its competitive ratio is bounded by c.

Previous Work. The problem of the present paper has been studied by many researchers [8, 3, 4]. Goldwasser [4] showed that the optimal competitiveness is $2 + \frac{1}{k}$, where the slack of each job is at least k times the processing time. Very recently, Lee [6] presented an $O(log\frac{1}{k})$-competitive randomized algorithm for a single machine and for multiple machines. There are also researches on several variants of this problem such as preemptive scheduling with different objectives [1, 5] and scheduling with abort [12, 7].

Team Adversary. Many researches have analyzed online algorithms using the competitive analysis [10]. It is assumed that the input of online problem is generated by the malicious adversary having unlimited computing power. Sometimes, however, the worst-case input request is too severe to the online algorithm and rare in practice. Thus, many researches studied various relaxed settings of online problems by incorporating lookahead, access graphs, stochastic adversary, or diffuse adversary (for details, see [2]).

In this paper we first introduce a quite natural parameterization of the adversary, called *team adversary*, which appears to be useful for analyzing the online scheduling problems with deadlines. The team adversary consists of m ($\in \mathbb{Z}^+$) weak adversaries and each weak adversary can generate *serial jobs* whose available intervals between the release time and the deadline are disjoint. Notice that the traditional adversary corresponds to the case of $m = \infty$.

From the practical viewpoint, we can view that the input generated by the m-team adversary comes from m different sources, each of which is *not* overloaded.

In addition to the analysis of the optimal competitiveness, there is another interesting question related with the m-team adversary: what is the smallest number m such that the m-team adversary can generate the worst-case input of the ∞-team adversary (i.e., the traditional adversary)? We give a tight answer for this question.

Restart. We consider the online algorithm with an additional capability. Our motivation is that the optimal competitiveness of $2+1/k$ given in [4] still goes to infinity as k goes to 0. We assume that online algorithms can abort the currently-running job in favor of better jobs and *restart* it *from scratch* later if its deadline can be met. Notice that this scheduler is different from the *preemptive* scheduler that can suspend the running job and restart it from the point of suspension. Moreover, the optimal algorithm is the same as the optimal algorithm without

abort/restart, because abort and restart would not increase the gain of any offline algorithm[1]. This model was mentioned in [9] and studied in different settings [7, 11]. Our framework is a generalization of Woeginger's [12], in which every job has no slack time and thus restart is not considered. In our framework, each job has non-zero slack time and can restart.

Our Results. We show that the optimal competitiveness against the team adversary of size m is $1 + \min[\frac{m-1}{k}, \frac{k+1}{k}]$, which is a generalization of the optimal competitiveness of $1 + \frac{k+1}{k}$ for the traditional adversary [4]. Our analysis implies that the m-team adversary has the same power as the traditional adversary (with respect to the competitive analysis) if and only if m is at least $k + 2$.

We next present a constant competitive strategy with abort/restart. The interesting characteristic of our algorithm is that it is parameterized by c, indicating *reluctance to abort*, and the best value of c depends on the value of k. If c is large enough, our algorithm never aborts. Consequently, the analysis includes the analysis of the algorithm without abort/restart as a special case.

2 Optimal Competitiveness against the m-Team Adversary

In this section, we describe the optimal competitiveness against the m-team adversary.

Theorem 1 *The optimal competitiveness against the m-team adversary is* $1 + \min[\frac{m-1}{k}, \frac{k+1}{k}]$.

In the rest of this section, we show Theorem 1. The proof for the second term $1+\frac{k+1}{k}$ is exactly same as the proof in [4] but we include it here for completeness.

Proof of the Lower Bound. We denote a job $\mathcal{J}_j$ by $\langle r_j, p_j, s_j \rangle$, where s_j is assumed to be at least $k \cdot p_j$. Imagine we are the m-team adversary who would like to raise the competitive ratio of any online algorithm. Initially, one unit job $\mathcal{J}_1$ with arbitrarily large slack is given. As this job may be the only one to arrive, any online algorithm must schedule it. Suppose it was scheduled at t by the online algorithm. There are two cases.

- $\frac{m-1}{k} < \frac{k+1}{k}$.
  Assume that ϵ (> 0) is arbitrarily small. The m-team adversary generates $m - 1$ copies of a job $\langle t + \epsilon, \frac{1-2\epsilon}{k}, 1 - 2\epsilon \rangle$. Since the expiration time of each job is earlier than $t + 1$, the online algorithm cannot schedule any of them. However the optimal algorithm can schedule all of them by scheduling $\mathcal{J}_1$

[1] In the non-preemptive scheduling, it is said that once a job begins running, it is not interrupted. Interestingly, there are two ways to interpret and implement this definition. First, each job may be inherently non-preemptive. Second, only a non-preemptive execution of each job is regarded as a success. In the offline scheduling, there is no difference between the above two. In the online scheduling, however, there is a big difference, as shown in this paper.

elsewhere. The competitive ratio is at least $1+\frac{m-1}{k}-(m-1)2\epsilon/k$. By taking $\epsilon' = 2(m-1)\epsilon/k$, we have that the competitive ratio is at least $1+\frac{m-1}{k}-\epsilon'$, for any ϵ' (> 0).

– $\frac{m-1}{k} \geq \frac{k+1}{k}$.
The m-team adversary gives one job $\langle t+\epsilon, \frac{1-2\epsilon}{k}, 1-2\epsilon\rangle$ and $m-2$ copies of job $\langle t+\epsilon, \frac{1-2\epsilon}{m-2}, 1-2\epsilon\rangle$. Since the expiration time of each job is $t+1-\epsilon$, the online algorithm cannot schedule any of them. On the other hand, all jobs can be scheduled by the optimal algorithm. Thus the competitiveness is at least $1+\frac{k+1}{k}-2(1+\frac{1}{k})\epsilon$, for an arbitrarily small ϵ (> 0). By taking $\epsilon' = 2(1+\frac{1}{k})\epsilon$, we have that the competitive ratio is at least $1+\frac{k+1}{k}-\epsilon'$.

Proof of the Upper Bound. We say that a job $\mathcal{J}_j$ is *available* at $t \in [r_j, e_j]$ if $\mathcal{J}_j$ is neither be completed nor be running at t. We consider the online algorithm `EEF` (`Earliest Expiration-time First`) that schedules the available job with the earliest expiration time. We show that the competitive ratio of `EEF` is no greater than both of $1+\frac{m-1}{k}$ and $1+\frac{k+1}{k}$.

As in [3], we consider the schedule σ produced by `EEF`, and call the periods during which the resource is continuously in use the *busy periods* of σ. Label these periods as $\pi_1, \pi_2, \cdots, \pi_l$ and let b_i and e_i, respectively, denote the times at which π_i begins and ends. Partition the job sequence S into classes $S_1, S_2, \cdots, S_l$, where S_i consists of exactly those jobs that arrived during the period $[b_i, e_i)$. In order to prove that `EEF` is c-competitive, it suffices to prove this result over subsequence S_1. Without loss of generality, we can artificially push back the release times of all other jobs to be later than the latest deadline of a job in S_1. This change will have no effect on the gain for `EEF` and the additional time can only help the optimal offline algorithm. Therefore, from the point on we assume that our instances are such that the `EEF` schedule results in a single busy period. Let $[0, t)$ be this single busy period.

Let $\mathcal{I}_i$ denote the set of jobs generated by the i-th adversary. Suppose that jobs in A_i $(\subseteq \mathcal{I}_i)$ are scheduled by `EEF` and those in B_i $(\subseteq \mathcal{I}_i)$ are scheduled by the optimal algorithm `OPT` but not scheduled by `EEF`. Let a_i and b_i, respectively, denote the sum of the lengths of jobs in A_i and B_i. Then we have the following lemma.

Lemma 2 (a) $\sum_i b_i \leq t(1+\frac{1}{k})$. (b) $b_i \leq \frac{t-a}{k}$.

Proof: (a) We first note that `OPT` cannot start to schedule any job in B_i after t, because otherwise, `EEF` can also schedule it after t, which contradicts the definition of t. Next, it is easily seen that all jobs in A_i and B_i arrive after 0 because `EEF` schedules a job at 0 for the first time. Finally, we claim that every job in B_i is shorter than $\frac{t}{k}$. Indeed otherwise, the slack of such a job is at least t and thus it must be available after t, so `EEF` will schedule at least one job after t, which is a contradiction. Thus `OPT` starts to schedule jobs in B_i during $[0, t)$ and the last scheduled job can be no larger than $\frac{t}{k}$. Therefore $\sum_i b_i$ is at most $t(1+\frac{1}{k})$.
(b) We fix i. Let $A_i \cup B_i = \{\mathcal{J}_{i_1}, \mathcal{J}_{i_2}, \cdots, \mathcal{J}_{i}\ \}$, where $r_{i_1} \leq r_{i_2} \leq \cdots \leq r_{i}$. Recall that A_i is the set of jobs scheduled by `EEF` and any job in B_i is not available

at the time that some job in A_i is scheduled, because they are generated by the same (i-th) adversary. It is easily seen that $r_{i_1} \geq 0$ and $r_i \geq \sum_{l=1}^{k-1} p_i\,(1+k)$. Moreover, the expiration time e_i is at least $r_i + p_i \cdot k$. So the last job's expiration time is $e_i \geq \sum_{l=1}^{n-1} p_i + (a_i + b_i) \cdot k$. Assume that the last job belongs to B_i. Then $\sum_{l=1}^{n-1} p_i \geq a_i$. If this lemma does not hold (i.e., $b_i > \frac{t-a}{k}$),

$$\begin{aligned} e_i &\geq \sum_{l=1}^{n-1} p_i + (a_i + b_i) \cdot k \\ &\geq a_i + b_i \cdot k \\ &> a_i + t - a_i \\ &= t \end{aligned}$$

Since the expiration time of the last job in B_i is later than t, this job is available after t and thus the online algorithm `EEF` must schedule it after t (or other jobs in B_i instead), which contradicts the definition of t.

Now assume that the last job belongs to A_i. Then $\sum_{l=1}^{n-1} p_i \geq b_i$. The execution of this job cannot end before $r_i + p_i$. If this lemma does not hold (i.e., $b_i > \frac{t-a}{k}$),

$$\begin{aligned} r_i + p_i &\geq \sum_{l=1}^{n-1} p_i\,(1+k) + p_i \\ &= a_i + b_i + \sum_{l=1}^{n-1} p_i \cdot k \\ &\geq a_i + b_i + b_i \cdot k \\ &> a_i + b_i + t - a_i \\ &\geq t \end{aligned}$$

We have that the last job, scheduled by `EEF`, cannot finish before t, which is a contradiction to the definition of t. Therefore, this lemma holds. □

The gain of the optimal algorithm is at most $\sum_i a_i + \sum_i b_i$ and the gain of the online algorithm is t. The competitive ratio of `EEF` is $\frac{\sum a + \sum b}{t}$. Plugging the bound of Lemma 2a and noting $t = \sum_i a_i$, we have that the competitiveness is at most $1 + \frac{k+1}{k}$. Plugging the bound of Lemma 2b, we have that

$$\begin{aligned} \frac{\sum_i a_i + \sum_i b_i}{t} &\leq \frac{\sum_i a_i + \sum_i \frac{t-a}{k}}{t} \\ &= \frac{t + \frac{(m-1)t}{k}}{t} \\ &= 1 + \frac{m-1}{k} \end{aligned}$$

completing the proof of Theorem 1.

3 Online Algorithm with Abort and Restart

In this section, we consider an online algorithm with an additional capability. That is, an online algorithm is allowed to *abort* the current running job in favor of better jobs and *restart* it *from scratch* later (if the deadline does not expire). We emphasize that this scheduler is different from the *preemptive* scheduler that can suspend the running job and resume it from the point of suspension. Moreover, the optimal algorithm is same as in the previous section, because abort and restart would not increase the gain of any offline algorithm.

We introduce a variant of `EEF` with the parameter c (> 1), called `EEF-c`. Let $||\cdot||$ denote a length of a job and let A denote the set of *available* jobs that are neither scheduled nor expired so far. Whenever the resource is idle, `EEF-c` first chooses a job $\mathcal{J}_j$ by `EEF`. The only one difference from `EEF` is that during the execution of the current job $\mathcal{J}_j$, if there is an available job $\mathcal{J}_i$ such that $||\mathcal{J}_i|| > c \cdot ||\mathcal{J}_j||$ and J_i will expire before the completion of the current execution of $\mathcal{J}_j$, `EEF-c` aborts $\mathcal{J}_j$ and schedules $\mathcal{J}_i$. The aborted job $\mathcal{J}_j$ is put into A if it does not expire. Observe that if c is sufficiently large `EEF-c` is same as `EEF`.

Theorem 3 *The competitive ratio of* `EEF-c`*, against the m-team adversary, is bounded by the followings, for any $m \in Z^+$ and c (> 1):*

$$\leq (\frac{\max(1-ck,0)}{c-1}+1)(\frac{m-1}{k})+1 \quad for\ k>0 \tag{1}$$

$$\leq (\frac{\max(1-ck,0)}{c-1}+1)(\frac{k+1}{k})+1 \quad for\ k>0 \tag{2}$$

$$\leq \frac{\max(1-ck,0)}{c-1}+1+c+1 \qquad for\ k\geq 0 \tag{3}$$

Proof: Using the arguments in the proof of Theorem 1, we can assume that our instances are such that the `EEF-c` schedule results in a single busy period $[0,t)$. That is, the resource is continuously in use during $[0,t)$ by `EEF-c` and it is idle right after t.

It is possible that one job is aborted several times and accepted later. Throughout this proof, a task refers to an (possibly partial) execution instance of a job. A job is *accepted* if one of its tasks is accepted. By the algorithm `EEF-c`, at most one task is accepted for each job. Thus the sum of the lengths of the accepted tasks is that of the accepted jobs.

Since a scheduled task may be aborted, the gain of `EEF-c` (over $[0,t)$) may be smaller than t. Let us first bound the gain of `EEF-c` from below. Assume that $\mathcal{J}_{i_1}, \mathcal{J}_{i_2}, \cdots, \mathcal{J}_{i_l}$ are accepted tasks and each of $\mathcal{J}_{i_k}$ $(1 \leq k \leq l)$ completes its execution during $[a_{i_k}, b_{i_k})$. Let $\mathcal{J}^k_{j_1}, \mathcal{J}^k_{j_2}, \cdots, \mathcal{J}^k_{j_{l'}}$ denote the tasks partially scheduled in this order during $[b_{i_{k-1}}, a_{i_k})$ and aborted before completion. We claim that

$$a_{i_k} - b_{i_{k-1}} \leq \frac{\max(1-ck,0)}{c-1} \cdot (b_{i_k} - a_{i_k}) \tag{4}$$

To show this claim, we first fix our attention to two tasks $\mathcal{J}_j^k{}_,$ and $\mathcal{J}_i$. The former was aborted at a_i in order to schedule the latter. From the definition of `EEF-c`, we have $||\mathcal{J}_i\ || > c \cdot ||\mathcal{J}_j^k{}_,||$. Similarly, $||\mathcal{J}_j^k{}_{+1}|| > c \cdot ||\mathcal{J}_j^k\,||$ $(1 \le r < k')$. Thus

$$\sum_{r=1}^{k'} ||\mathcal{J}_j^k\,|| < \frac{1}{c-1}||\mathcal{J}_i\ ||$$

Again consider $\mathcal{J}_j^k{}_,$ and $\mathcal{J}_i$. Let $t_{k'}$ denote the time at which $\mathcal{J}_j^k{}_,$ is partially scheduled. If $\mathcal{J}_i$ is available at $t_{k'}$, $\mathcal{J}_j^k{}_,$ never runs (in other words, $t_{k'} = a_i$) and so it contributes nothing to the left part of (4) and can be ignored. Assume otherwise, that is, $\mathcal{J}_i$ arrives after $t_{k'}$ and $\mathcal{J}_j^k{}_,$ is executed for some time. Since $\mathcal{J}_j^k{}_,$ is eventually aborted, $\mathcal{J}_i$ expires before $\mathcal{J}_j^k{}_,$ completes its execution. Since the slack of $\mathcal{J}_i$ is at least $k \cdot ||\mathcal{J}_i\ || > ck||\mathcal{J}_j^k{}_,||$, the length of the partial execution of $\mathcal{J}_j^k{}_,$ is at most $||\mathcal{J}_j^k{}_,|| - ck||\mathcal{J}_j^k{}_,|| = (1-ck)||\mathcal{J}_j^k{}_,||$. Plugging this formula into (4), we have that

$$a_i\ - b_i\ {}_{-1} \le \max(1-ck,0) \cdot \sum_{r=1}^{k'} ||\mathcal{J}_j^k\,|| < \frac{\max(1-ck,0)}{c-1}||\mathcal{J}_i\ || \qquad (5)$$

Summing the equation (5) over all accepted jobs $\mathcal{J}_{i_1}, \mathcal{J}_{i_2}, \cdots, \mathcal{J}_i$, we have

$$t < [\frac{\max(1-ck,0)}{c-1} + 1] \cdot \sum_{r=1}^{l} ||\mathcal{J}_i\ || \qquad (6)$$

Clearly the gain of `EEF-c` is $\sum_{r=1}^{l} ||\mathcal{J}_i\ ||$.

How much is the gain of the optimal algorithm? The optimal algorithm `Opt`, at best, can fill the busy period $[0,t)$, start a new job right before t and schedule the jobs accepted by `EEF-c` outside $[0,t)$. By applying the arguments in the proof of Theorem 1, we have the the following upper bound on the gain of `Opt`:

$$\begin{aligned} Gain(\texttt{Opt}) &\le \min(\frac{m-1}{k}, \frac{k+1}{k})t + \sum_{r=1}^{l} ||\mathcal{J}_i\ || \\ &\le [\min(\frac{m-1}{k}, \frac{k+1}{k})(\frac{\max(1-ck,0)}{c-1} + 1) + 1] \sum_{r=1}^{l} ||\mathcal{J}_i\ || \\ &= [\min(\frac{m-1}{k}, \frac{k+1}{k})(\frac{\max(1-ck,0)}{c-1} + 1) + 1] \cdot Gain(\texttt{EEF-c}) \end{aligned}$$

completing the proof of (1) and (2).

It remains to show (3). Recall that the optimal algorithm, at best, can fill the busy period $[0,t)$, start a new job $\mathcal{J}$ right before t and schedule the jobs accepted by `EEF-c` outside $[0,t)$. It is easily seen that $||\mathcal{J}|| \le c \cdot ||\mathcal{J}_i\,||$. Indeed otherwise, `EEF-c` either aborts $\mathcal{J}_i$ and schedules $\mathcal{J}$ instead (if $\mathcal{J}$ expires before t) or schedules $\mathcal{J}$ (possibly other jobs instead) after t (if $\mathcal{J}$ does not expire before t), which is a contradiction of the definition of $\mathcal{J}_i$ and $[0,t)$, respectively.

Therefore we have the the following upper bound on the gain of `Opt`:

$$\begin{aligned} Gain(\texttt{Opt}) &\leq t + c \cdot ||J_i|| + \sum_{k=1}^{l} ||\mathcal{J}_i|| \\ &\leq (\frac{\max(1-ck, 0)}{c-1} + 1 + c + 1] \cdot Gain(\texttt{EEF-c}) \end{aligned}$$

completing the proof of this theorem. □

How good is the bound in Theorem 3? First of all, the competitive ratio is constant for any value k (> 0). If we let $c = 2$, the bound in (3) becomes $4 + 1 - ck$ (≤ 5). The best competitive ratio of `EEF-c` depends on the choice of c, which again depends on the value of k. Clearly the best competitive ratio of `EEF-c` is no greater than that of `EEF`. It is more difficult to prove the lower bound for the online algorithm with abort and restart, because the behavior of the online algorithm is more complex. However the bound in Theorem 3 is not far from the lower bound because the lower bound of 4 is known in [12] for the special case of this problem where all jobs have no slack time.

References

1. S. Baruah, J. Harita, and X. Sharma. Online scheduling to maximize task completions. In *Proc. of IEEE Real-Time Systems Symposium*, pages 228–237, 1994.
2. A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.
3. S. Goldman, J. Parwatikar, and S. Suri. On-line scheduling with hard deadlines. *Journal of Algorith*, 34(2):370–389, 2000.
4. M. Goldwasser. Patience is a virtue: the effect of slack on competitiveness for admission control. In *Proc. of 10th ACM-SIAM SODA*, pages 396–405, 1999.
5. B. Kalyanasundaram and K. Pruhs. Maximizing jon completion online. In *Proc. of ESA*, pages 235–246, 1998.
6. Jae-Ha Lee. Online deadline scheduling: multiple machines and randomization. In *Proc. of SPAA*, pages 21–25, 2003.
7. Jae-Ha Lee and Kyung-Yong Chwa. Online scheduling of parallel communications with individual deadlines. In *Proc. of ISAAC*, pages 383–392, 1999.
8. R.J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. of 5-th ACM SODA*, pages 302–305, 1994.
9. J. Sgall. Online scheduling. *In Online Algorithms: The State of the Art, eds. A. Fiat and G. J. Woeginger, LNCS 1442, Springer Verlag*, pages 196–231, 1998.
10. D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of ACM*, 2(28):202–208, 1985.
11. M. van den Akker, H. Hoogeven, and N. Vakhania. Restarts can help in the on-line minimization of the maximum delivery time on a single machine. In *Proc. of 8th ESA*, pages 427–436, 2000.
12. G.J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130:5–16, 1994.

Minimum Sum Multicoloring on the Edges of Trees*
(Extended Abstract)

Dániel Marx

Department of Computer Science and Information Theory,
Budapest University of Technology and Economics
Budapest Pf. 91, H-1521, Hungary
dmarx@cs.bme.hu

Abstract. The edge multicoloring problem is that given a graph G and integer demands $x(e)$ for every edge e, assign a set of $x(e)$ colors to vertex e, such that adjacent edges have disjoint sets of colors. In the *minimum sum edge multicoloring problem* the *finish time* of an edge is defined to be the highest color assigned to it. The goal is to minimize the sum of the finish times. The main result of the paper is a polynomial time approximation scheme for minimum sum multicoloring the edges of trees.

1 Introduction

In this paper we study an edge multicoloring problem that is motivated by applications in scheduling. We are given a graph with an integer demand $x(e)$ for each edge e. A *multicoloring* is an assignment of a set of $x(e)$ colors to each edge e such that the colors assigned to adjacent edges are disjoint. In multicoloring problems the usual aim is to minimize the total number of different colors used in the coloring. However, in this paper a different optimization goal is studied. Given a multicoloring, the *finish time* of an edge is defined to be the highest color assigned to it. In the minimum sum multicoloring problem the goal is to minimize the sum of finish times.

An application of edge coloring is to model dedicated scheduling of biprocessor tasks. The vertices correspond to the processors and each edge $e = uv$ corresponds to a job that requires $x(e)$ time units of simultaneous work on the two preassigned processors u and v. The colors correspond to the available time slots: by assigning $x(e)$ colors to edge e, we select the $x(e)$ time units when the job corresponding to e is executed. A processor cannot work on two jobs at the same time, this corresponds to the requirement that a color can appear at most once on the edges incident to a vertex. The finish time of edge e corresponds to the time slot when job e is finished, therefore minimizing the sum of the finish

* Research is supported in part by grants OTKA 44733, 42559 and 42706 of the Hungarian National Science Fund.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 214–226, 2004.
© Springer-Verlag Berlin Heidelberg 2004

times is the same as minimizing the sum of completion times of the jobs. Using the terminology of scheduling theory, we minimize the mean flow time, which is a well-studied optimization goal in the scheduling literature. Such biprocessor tasks arise when we want to schedule file transfers between processors [2] or the mutual diagnostic testing of processors [6]. Note the we allow that a job is interrupted and continued later: the set of colors assigned to an edge does not have to be consecutive, hence our problem models preemptive scheduling.

Of particular interest is the case where the graph G is bipartite. A possible application of the bipartite problem is the following. One bipartition class corresponds to a set of clients, the other class corresponds to a set of servers. An edge e between two vertices means that the given client has to access the given server for $x(e)$ units of time. A client can access only one server at the same time, and a server cannot accept connections from more than one client simultaneously. Clearly, bipartite edge multicoloring models this situation.

Minimum sum edge multicoloring is **NP**-hard on bipartite graphs even if every edge has unit demand [3]. For general demands, [1] gives a 2-approximation algorithm. The problem can be solved in polynomial time if every edge has unit demand and the graph is a tree [3, 7].

In this paper, we consider the minimum sum edge coloring problem restricted to trees. We show that, unlike in the unit demand case, minimum sum edge coloring is **NP**-hard on trees if the demands are allowed to be at most 2. The main contribution of the paper is a polynomial time approximation scheme (PTAS) for minimum sum edge multicoloring in trees.

In [4, 5] PTAS is given for the vertex coloring version of the problem in the case when the graph is a tree, partial k-tree, or a planar graph. One of their main tools is the decomposition of colors into layers of geometrically increasing size. This method will be used in this paper as well. However, most of the other tools in [4, 5] cannot be applied in our case, since those tools assume that the graph can be colored with a constant number of colors. If the maximum degree of the tree can be arbitrary, then the line graph of the tree can contain arbitrarily large cliques. On one hand, these large cliques make the tools developed for partial k-trees impossible or very difficult to apply. On the other hand, a large clique helps us in finding an approximate solution: since the sum of a large clique has to be very large in every coloring (it grows quadratically in the size of the clique), more errors can be tolerated in an approximate solution, and this gives us more elbow space in constructing a good approximation.

The paper is organized as follows. In Section 2 we introduce some notations and give results on the complexity of the problem. Section 3 gives a PTAS for trees where the maximum degree of the tree is bounded by a constant, while Section 4 gives a linear time PTAS for general trees.

2 Preliminaries

The problem considered in this paper is the edge coloring version of minimum sum multicoloring, which can be stated formally as follows:

Minimum Sum Edge Multicoloring (semc)
Input: A graph $G(V, E)$ and a *demand* function x: $E \to \mathbb{N}$.
Output: A *multicoloring* Ψ: $E \to 2^{\mathbb{N}}$ such that $|\Psi(e)| = x(e)$ for every edge e, and $\Psi(e_1) \cap \Psi(e_2) = \emptyset$ if e_1 and e_2 are adjacent in G.
Goal: The *finish time* of edge e in coloring Ψ is the highest color assigned to it, $f_\Psi(e) = \max\{c \in \Psi(e)\}$. The goal is to minimize $\sum_{e \in E} f_\Psi(e)$, the *sum* of the coloring Ψ.

We extend the notion of finish time to a set E of edges by defining $f_\Psi(E) = \sum_{e \in E} f_\Psi(e)$. Given a graph G and a demand function $x(e)$ on the edges of G, the minimum sum that can be obtained is denoted by $\mathrm{OPT}(G, x)$.

Henceforth the graph G is a rooted tree with root r. The root is assumed to be a node of degree one, the *root edge* is the edge incident to r. Every edge has an *upper node* (closer to r) and a *lower node* (farther from r). Edge f is a *child edge* of edge e if the upper node of f is the same as the lower node of e. In this case, edge e is the *parent edge* of edge f. A node is a *leaf node* if it has no children, and an edge is a *leaf edge* if its lower node is a leaf node. The subtree T_e consists of the edge e and the subtree rooted at the lower node of e.

A *bottom up traversal* of the edges is an ordering of the edges in such a way that every edge appears before its parent edge. It is clear that such ordering exists and can be found in linear time.

Since the tree is bipartite, every node has a *parity*, which is either 1 or 2, and neighboring nodes have different parity. Let the parity of an edge be the parity of its upper node. Thus if two edges have the same parity and they have a common node v, then v is the upper node of both edges.

If the tree has maximum degree Δ, then the edges can be colored with Δ colors. This color will be called the *type* of the edge. In some of the algorithms, the leaf edges are special, they are handled differently, therefore we want to assign a type only to the non-leaf edges. Clearly, if every edge has at most D non-leaf child edges, then the non-leaf edges can be colored with $D + 1$ colors, and they can be given a type from $1, 2, \ldots, D + 1$ such that adjacent edges have different type.

The following lemma bounds the number of colors required in a minimum sum multicoloring (proof is omitted). In the following, the maximum demand in the instance is denoted by p.

Lemma 1. *If T is a tree with maximum degree Δ and maximum demand p, then every optimum coloring of the* **semc** *problem uses at most $p(2\Delta - 1)$ colors.*

If both the maximum degree of the tree and the maximum demand is bounded by a constant, then the problem can be solved in linear time. The idea is that there are only a constant number of possible color sets that can appear at each edge, hence using standard dynamic programming techniques, the optimum coloring can be found during a bottom up traversal of the edges. We omit the details.

Theorem 2. *The* **semc** *problem for trees can be solved in $2^{O(p\Delta)} \cdot n$ time.*

On the other hand, if only the demand is bounded, then the problem becomes **NP**-complete:

Theorem 3. *Minimum sum edge coloring is* **NP***-complete in trees, even if every demand is 1 or 2.*

Using polyhedral techniques, we can show that trees have the following scaling property (proof is omitted):

Theorem 4. *For every tree T, demand function x and integer q, if $x'(e) = q \cdot x(e)$, then $\mathrm{OPT}(T, x') = q \cdot \mathrm{OPT}(T, x)$ holds.*

In general, Theorem 4 does not hold for every graph, not even for every bipartite graph. We will use Theorem 4 to reduce the number of different demand sizes that appear in the graph, with only a small increase of the sum:

Lemma 5. *Let (T, x) be an instance of* **semc** *and let $x'(e)$ be $\lfloor (1+\epsilon)^i \rfloor$ for the smallest i such that $\lfloor (1+\epsilon)^i \rfloor \geq x(e)$. Then $\mathrm{OPT}(T, x') \leq (1+\epsilon) \cdot \mathrm{OPT}(T, x)$.*

As explained in the introduction, our goal is to minimize the sum of finish times, not to minimize the number of different colors used. Nevertheless, in Theorem 6 we show that the minimum number of colors required for the coloring the edges of a tree can be determined in polynomial time. The approximation algorithm presented in Section 3 uses this result to solve certain subproblems.

Theorem 6. *Let T be a tree and let $C = \max_{v \in V(T)} \sum_{e \ni v} x(e)$. Every coloring of T uses at least C colors, and one can find in linear time a multicoloring Ψ using C colors where each $\Psi(e)$ consists of at most two intervals of colors. Moreover, if each $x(e)$ is an integer multiple of some integer q, then we can find such a Ψ where the intervals in each $\Psi(e)$ are of the form $[qi_1 + 1, qi_2]$ for some integers i_1 and i_2.*

Proof. It is clear that at least C colors are required in every coloring: there is a vertex v such that the edges incident to v require C different colors. The coloring Ψ can be constructed by a simple greedy algorithm. Details omitted. □

3 Bounded Degree

If a tree T has maximum degree Δ, then the line graph of T is a partial $(\Delta - 1)$-tree. Halldórsson and Kortsarz [4] gave a PTAS with running time $n^{O(k^2/\epsilon^2)}$ for minimum sum multicoloring the vertices of partial k-trees, therefore there is a PTAS for **semc** in bounded degree trees as well. However, the method can be made simpler and more efficient in line graphs of trees. In this section we present a linear time PTAS for **semc** in bounded degree trees, which makes use of the special structure of trees. Furthermore, our algorithm works even if the degree of the tree is not bounded, but we know that every edge has a bounded number of non-leaf child edges. Most of the ideas presented in this section are taken from [4], with appropriate modifications. In Section 4 a PTAS is given for general trees, which uses the result in this section as a subroutine.

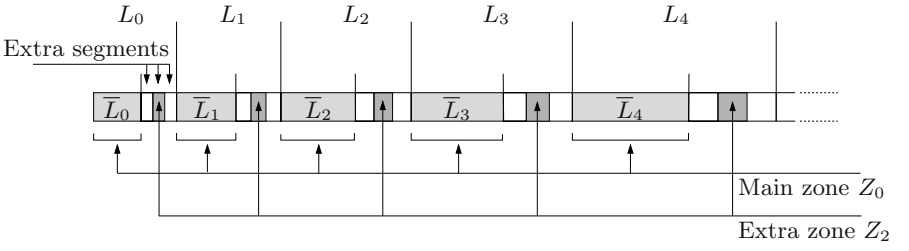


Fig. 1. The decomposition of the colors into layers ($\ell = 3$)

3.1 Layers and Zones

An important idea of the approximation schemes given in [4, 5] is to divide the color spectrum into geometrically increasing layers, and to solve the problem in these layers separately. We use a similar method for the **semc** problem in bounded degree trees (Theorem 9) and general trees (Theorem 10).

For some $\epsilon > 0$ and integer $\ell \geq 0$, the (ϵ, ℓ)-*decomposition* divides the set of colors into *layers* $L_0, L_1, \ldots$ and *zones* $Z_0, Z_1, \ldots, Z_\ell$. The layers are of geometrically increasing size: layer L_i contains the range of colors from q_i to $q_{i+1} - 1$, where $q_i = \lfloor (1+\epsilon)^i \rfloor$ (if $q_i = q_{i+1}$, then layer L_i is empty). Denote by $Q_i^\epsilon = |L_i| = q_{i+1} - q_i$ the size of the ith layer. Notice that $q_{i+1} \leq (1+\epsilon)q_i$. The total size of layers L_0, L_1, ..., L_i is $q_{i+1} - 1$.

Each layer is divided into a main block and an extra block. The extra block is further divided into extra segments (see Figure 1). Layer L_i is divided into two parts: the first $\frac{1}{1+\epsilon\ell} Q_i^\epsilon$ colors form the *main block* of layer L_i and the remaining $\frac{\epsilon\ell}{1+\epsilon\ell} Q_i^\epsilon$ colors the *extra block*. The main block of layer L_i is denoted by $\overline{L}_i$. The union of the main block of every layer L_i is the *main zone* Z_0. Divide the extra block of every layer L_i into ℓ equal parts: these are the ℓ *extra segments* of L_i. The union of the jth extra segment of every layer L_i forms the jth extra zone Z_j. Each extra zone contains $\frac{\epsilon}{1+\epsilon\ell} Q_i^\epsilon$ colors from layer L_i. Rounding problems are not discussed in this extended abstract, but they can be handled rigorously.

We will need the following properties of the defined zones:

Lemma 7. *For given ϵ and ℓ, the (ϵ, ℓ)-decomposition of the colors has the following properties:*

(a) For every $c \geq 1$, there are at least c colors in Z_0 not greater than $\lfloor (1+\epsilon\ell)c \rfloor$.
(b) For every $c \geq 1$ and $1 \leq j \leq \ell$, there are at least c colors in Z_j not greater than $\lceil (1+\epsilon)(1+\epsilon\ell)/\epsilon \cdot c \rceil$.

Given a multicoloring Ψ, the operation (ϵ, ℓ)-*augmentation* creates a multicoloring Φ in the following way. Consider the (ϵ, ℓ)-decomposition of the colors, and if $\Psi(e)$ contains color c, then let $\Phi(e)$ contain instead the cth color from the main zone Z_0. By Lemma 7a, $f_\Phi(e) \leq \lfloor (1+\epsilon\ell) f_\Psi(e) \rfloor$, thus this operation increases the sum by at most a factor of $(1+\epsilon\ell)$. After the augmentation, the colors of the extra zones are not used, only the colors of the main zone.

3.2 PTAS for Bounded Degree Trees

The polynomial time algorithm of Theorem 2 was based on the observation that we have to consider only a constant number of different colorings at each edge if both the demand and the maximum degree is bounded. In general, however, the number of different color sets that can be assigned to an edge is exponentional in the demand. The following lemma shows that we can find a good approximate coloring by considering only a restricted type of color sets.

Lemma 8. *For $\epsilon > 0$, define $b_{i,j} = q_i + j\lceil \epsilon^2 Q_i^\epsilon/4 \rceil$. If each vertex of the tree T has at most D non-leaf child edges, then it has an $(1+\epsilon)(1+(D+1)\epsilon)$-approximate coloring Ψ with the following properties:*

1. *In the (ϵ, D)-decomposition of the colors, if e is a non-leaf edge, then $\Psi(e)$ contains colors from the main zone only between $\epsilon x(e)/4$ and $2x(e)/\epsilon$.*
2. *If e is a non-leaf edge of type k, then $\Psi(e)$ contains the first t_e colors from extra zone Z_k (for some t_e), and it does not contain colors from the other extra zones.*
3. *If e is a leaf edge, then $\Psi(e)$ contains colors only from the main zone.*
4. *If e is a non-leaf edge, then $\Psi(e)$ contains at most two continuous intervals of colors from the main block of each layer, and the intervals in layer L_i are of the form $[b_{i,j_1}, b_{i,j_2} - 1]$ for some j_1 and j_2.*

Proof. Let Φ be an optimum solution, and let Ψ be the result of an (ϵ, D)-augmentation on Φ. By Lemma 7a, $f_\Psi(e) \leq (1 + (D + 1))f_\Phi(e)$ for every e.

If $f_\Psi(e) > 2x(e)/\epsilon$ for a non-leaf edge e of type j, then modify $\Psi(e)$ to be the the first $x(e)$ colors of zone Z_j. By Lemma 7b, Z_j contains at least $x(e)$ colors not greater than $\lceil (1+\epsilon)(1+(D+1)\epsilon)/\epsilon \cdot x(e) \rceil \leq 2(1+(D+1)\epsilon)x(e)/\epsilon$. Therefore the $x(e)$ colors assigned to e are not greater than $2(1 + (D + 1)\epsilon)x(e)/\epsilon$, hence $f_\Psi(e) \leq (1 + (D + 1))f_\Phi(e)$.

If $\Psi(e)$ contains colors in the main zone below $\epsilon x(e)/4$, then delete these colors and let $\Psi(e)$ contain instead the first $\epsilon x(e)/4$ colors from zone Z_j. There are at least $\epsilon x(e)/4$ colors in Z_i below $\lceil (1 + \epsilon)(1 + (D + 1)\epsilon)/\epsilon \cdot \epsilon x(e)/4 \rceil \leq (1 + (D + 1))x(e) \leq (1 + (D + 1))f_\Phi(e)$. Therefore $f_\Psi(e) \leq (1 + (D + 1))f_\Phi(e)$ for each edge e, and Ψ is an $(1 + (D + 1)\epsilon)$-approximate solution satisfying the first three properties of the lemma.

Finally, we make Ψ satisfy the fourth requirement as well. For each non-leaf edge e, let $x_i(e)$ be $|\Psi(e) \cap \overline{L}_i|$ rounded down to the next integer multiple of $\lceil \epsilon^2 Q_i^\epsilon/4 \rceil$. If we use x_i as a demand function on the non-leaf edges of the tree, then there is multicoloring satisfying x_i that uses at most $|\overline{L}_i|$ colors: $\Psi(e) \cap \overline{L}_i$ is such a coloring. Therefore by Theorem 6, there is a multicoloring Ψ_i that uses at most $|\overline{L}_i|$ colors, satisfies x_i, and where each $\Psi(e)$ consists of at most two intervals of the form $[1 + j_1\lceil \epsilon^2 Q_i^\epsilon/4 \rceil, j_2\lceil \epsilon^2 Q_i^\epsilon/4 \rceil]$ for some j_1, j_2. Modify coloring Ψ: let Ψ_i determine how the colors are assigned in the main zone of layer i. Now the third requirement is satisfied, but it is possible that Ψ assigns less than $x(e)$ colors to an edge. We can lose at most $\lceil \epsilon^2 Q_i^\epsilon/4 \rceil - 1 \leq \epsilon^2 Q_i^\epsilon/4$ colors in layer i, hence we lose at most the $\epsilon^2/4$ part of each layer. Since $\Psi(e)$ contains

colors only up to $2x(e)/\epsilon$, thus we lose only at most $\epsilon^2/4 \cdot 2x(e)/\epsilon = \epsilon x(e)/2$ colors. If non-leaf edge e is of type j, then we use extra zone Z_j to replace the lost colors. So far, edge e uses at most $\epsilon x(e)/4$ colors from Z_j (previous paragraph), hence there are still place for more than $3\epsilon x(e)/4$ colors in Z_j below $(1 + (D + 1)\epsilon)x(e) \leq (1 + (D + 1)\epsilon)f_\Phi(e)$.

The modification in the previous paragraph can change the finish times of the non-leaf edges, but the largest color of each edge remains in the same layer, hence the finish time can increase by at most a factor of $1 + \epsilon$. Moreover, since we modified only the non-leaf edges, there can be conflicts between the non-leaf and the leaf edges. But that problem is easy to solve: since the number of colors used by the non-leaf edges at vertex v from layer i does not increase, there are enough colors in layer i for the leaf edges. The largest color of each leaf edge will be in the same layer, hence its finish time increases by at most a factor of $1 + \epsilon$, and $f_\Psi(e) \leq (1 + \epsilon)(1 + (D + 1)\epsilon)f_\Phi(e)$ follows for every edge e. □

Call a coloring satisfying the requirements of Lemma 8 a *standard* coloring. Notice that on a non-leaf edge e only a constant number of different color sets can appear in standard colorings: the main zone is not empty only in a constant number of layers, and in each layer the (at most two) intervals can be placed in a constant number of different ways. More precisely, in a standard coloring edge e can use the main zone only from layer $\log_{1+\epsilon} \epsilon x(e)/4$ to layer $\log_{1+\epsilon} 2x(e)/\epsilon$, that is, only in $\log_{1+\epsilon}((2x(e)/\epsilon)/(\epsilon x(e)/4)) = \log_{1+\epsilon} 8/\epsilon^2 = O(1/\epsilon \cdot \log 1/\epsilon)$ layers. In each layer, the end points of the intervals can take only at most $4/\epsilon^2$ different values, hence there are $(4/\epsilon^2)^2$ different possibilities for each of the two intervals. Therefore if we denote by $\mathcal{C}_e$ the different color sets that can appear in a standard coloring on non-leaf edge e, then $|\mathcal{C}_e| = ((4/\epsilon^2)^4)^{O((1/\epsilon)\cdot \log 1/\epsilon)} = 2^{O((1/\epsilon)\cdot \log^2 1/\epsilon)}$.

Theorem 9. *If every edge of $T(V, E)$ has at most D non-leaf child edges, then for every $\epsilon_0 > 0$, there is a $2^{O(D^2/\epsilon_0 \cdot \log^2(D/\epsilon_0))} \cdot n$ time algorithm that gives an $(1 + \epsilon_0)$-approximate solution to the* **semc** *problem.*

Proof. Set $\epsilon := \epsilon_0/(2D + 3)$. We use dynamic programming to find the best standard coloring: for every non-leaf edge e, and every set $S \in \mathcal{C}_e$, we determine $\mathrm{OPT}(e, S)$, which is defined to be the sum of the best standard coloring of T_e, with the additional requirement that edge e receives color set S. Clearly, if all the values $\{\mathrm{OPT}(r, S) \mid S \in \mathcal{C}_r\}$ are determined for the root edge r of T, then the minimum of these values is the sum of the best standard coloring, which is by Lemma 8 at most $(1 + \epsilon)(1 + (D + 1)\epsilon) \leq (1 + \epsilon_0)$ times the minimum sum.

The values $\mathrm{OPT}(e, S)$ are calculated in a bottom up traversal of the edges. Assume that e has k non-leaf child edges $e_1, e_2, \ldots, e_k$ and ℓ leaf child edges $e'_1, e'_2, \ldots, e'_\ell$. When $\mathrm{OPT}(e, S)$ is determined, the values $\mathrm{OPT}(e_i, S_i)$ are already available for every $1 \leq i \leq k$ and $S_i \in \mathcal{C}_e$. In a standard coloring of T_e every edge e_i is assigned a color set from $\mathcal{C}_e$. We enumerate all the $\prod_{i=1}^{k} |\mathcal{C}_e|$ possibilities for these color sets. For each combination $S_1 \in \mathcal{C}_{e_1}, \ldots, S_k \in \mathcal{C}_e$, we check whether these sets are pairwise disjoint. If so, then we determine the minimum sum that a standard coloring can have with these assignments. The minimum sum of subtree T_e with color set S_i on e_i is given by $\mathrm{OPT}(e_i, S_i)$. The finish

time of edge e can be calculated from S. Now only the leaf edges $e'_1, \ldots, e'_\ell$ remain to be colored. It is easy to see that the best thing to do is to order these leaf edges by increasing demand size, and color them one after the other, using the colors not already assigned to e, $e_1, \ldots, e_k$. Therefore we can calculate the minimum sum corresponding to a choice of color sets $S_1 \in \mathcal{C}_{e_1}, \ldots, S_k \in \mathcal{C}_e$, and we set $\mathrm{OPT}(e, S)$ to the minimum over all the combinations.

The algorithm solves at most $\sum_{e \in E} |\mathcal{C}_e| = n \cdot 2^{O((1/\epsilon)\cdot \log^2 1/\epsilon)}$ subproblems. To solve a subproblem, at most $2^{O(D\cdot(1/\epsilon)\cdot \log^2 1/\epsilon)}$ different combinations of the sets $S_1, \ldots, S_k$ have to be considered. Each color set can be described by $O(1/\epsilon \cdot \log 1/\epsilon)$ intervals, and the time required to handle each combination is polynomial in D and the number of intervals. Therefore the total running time of the algorithm is $2^{O(D\cdot 1/\epsilon \cdot \log^2(1/\epsilon))} \cdot n = 2^{O(D^2/\epsilon_0 \cdot \log^2(D/\epsilon_0))} \cdot n$. □

4 The General Case

In this section, we prove that **semc** admits a PTAS for arbitrary trees:

Theorem 10. *For every $\epsilon_0 > 0$, there is a $2^{O(1/\epsilon_0^{11} \cdot \log^2(1/\epsilon_0))} \cdot n$ time algorithm that gives an ϵ_0-approximate solution to the* **semc** *problem for every tree T and demand function x.*

Proof. Let $\epsilon := \epsilon_0/72$. The algorithm consists of a series of phases. The last phase produces a proper coloring of (T, x), and has cost at most $(1+\epsilon_0)\mathrm{OPT}(T, x)$. In the following we describe these phases.

Phase 1: Rounding the Demands. Using Lemma 5, we can assume that $x(e)$ is q_i for some i, modifying $x(e)$ this way increases the minimum sum by at most a factor of $1+\epsilon$. An edge e with demand q_i will be called a *class i* edge (if $x(e) = q_i$ for more than one i, then take the smallest i).

Phase 2: Partitioning the Tree. The edges of the tree are partitioned into subtrees in such a way that in a subtree the number of non-leaf child edges of a node is bounded by a constant. Now Theorem 9 can be used to find an approximate coloring for each subtree. These colorings can be merged into a coloring of the whole tree, but this coloring will not be necessarily a proper coloring, since there might be conflicts between edges that were in different subtrees. However, using a series of transformations, these conflicts will be resolved with only a small increase of the sum.

To obtain this partition, the edges of the tree are divided into *large edges* and *split edges*. It will be done in such a way that every node has at most $D := 4/\epsilon^5$ large child edges. If a node has less than D children, then its child edges are large edges. Let v be a node with at least D children, and denote by $n(v, i)$ the number of class i child edges of v. Let $N(v)$ be the largest i such that $n(v, i) > 0$ and set $F := 4/\epsilon^3$. Let e be a class i child edge of v. If $n(v, i) > F$, then e is a split edge, and it will be called a *frequent edge*. If $n(v, i) \leq F$ and $i \leq N(v) - \lfloor 1/\epsilon^2 \rfloor$, then

e is a split edge, and it will be a called a *small edge*. Otherwise, if $n(v,i) \leq F$ and $i > N(v) - \lfloor 1/\epsilon^2 \rfloor$, then e is a large edge. Clearly, v can have at most $F \cdot \lfloor 1/\epsilon^2 \rfloor = 4/\epsilon^3 \cdot \lfloor 1/\epsilon^2 \rfloor \leq 4/\epsilon^5 = D$ large child edges: for each class $N(v)$, $N(v)-1, \ldots, N(v) - \lfloor 1/\epsilon^2 \rfloor + 1$, there are at most F such edges.

The tree is split at the lower node of every split edge, the connected components of the resulting forest form the classes of the partition. Defined in another way: delete every split edge, make the connected components of the remaining graph the classes of the partition, and put every split edge into the class where its upper node belongs. Clearly, every split edge becomes a leaf edge in its subtree, thus if every node has at most D large child edges in the tree, then in every subtree every node has at most D non-leaf child edges.

Now assume that each subtree is colored with the algorithm of Theorem 9, this step can be done in $2^{O(D^2/\epsilon \cdot \log^2(D/\epsilon))} \cdot n = 2^{O(16/\epsilon^{10} \cdot 1/\epsilon \cdot \log^2(4/\epsilon^6))} \cdot n = 2^{O(1/\epsilon^{11} \cdot \log^2(1/\epsilon))} \cdot n$ time. Each coloring is an $(1+\epsilon)$-approximate coloring of the given subtree, thus merging these colorings yields a (not necessarily proper) coloring Ψ_1 of T such that $f_{\Psi_1}(T) \leq (1+\epsilon)OPT(T,x)$. In the rest of the proof, we transform Ψ_1 into a proper coloring in such a way that the sum of the coloring does not increase too much.

Phase 3: Small Edges. Consider the (ϵ, ℓ)-augmentation of the coloring Ψ_1 with $\ell := 6$. This results in a coloring Ψ_2 such that $f_{\Psi_2}(G) \leq f_{\Psi_1}(G)(1+\epsilon\ell)$ (see Section 3). First we modify Ψ_2 in such a way that the small edges use only the extra zones Z_1 and Z_2. More precisely, if a small edge e has parity $r \in \{1,2\}$, then e is recolored using the colors in Z_r (recall that the parity of the edge is the parity of its upper node). Since the extra zones contain only a very small fraction of the color spectrum, the recoloring can significantly increase the finish time of the small edges, roughly by a factor of $1/\epsilon$. However, we show that the total demand of the small edges are so small compared to the largest demand on the child edges of v, that their total finish time will be negligible, even after this large increase. By definition, the largest child edge of v has demand $q_{N(v)}$.

Consider the small edges whose upper node is v, a node with parity r. Color these edges one after the other, in the order of increasing demand size, using only the colors in Z_r. Call the resulting coloring Ψ_3. Let R_v be the set of small child edges of v. We claim that $f_{\Psi_3}(R_v) \leq \epsilon q_{N(v)}$ for every node v, thus transforming Ψ_2 into Ψ_3 increases the total sum by at most $\sum_{v \in T} f_{\Psi_3}(R_v) \leq \epsilon \sum_{v \in T} q_{N(v)} \leq \epsilon f_{\Psi_2}(T)$ and $f_{\Psi_3}(T) \leq (1+\epsilon) f_{\Psi_2}(T)$ follows. To give an upper bound on $f_{\Psi_3}(R_v)$, we assume the worst case, that is, $n(v,i) = F$ for every $i \leq N(v) - \lfloor 1/\epsilon^2 \rfloor$. Imagine first that the small edges are colored using the full color spectrum, not only with the colors of zone Z_r. Assume that the small edges are colored in the order of increasing demand size, and consider a class k edge e. In the coloring, only edges of class not greater than k are colored before e. Hence the finish time of e is at most

$$\sum_{i=0}^{k} n(v,i) q_i \leq F \sum_{i=0}^{k} (1+\epsilon)^i \leq 4(1+\epsilon)/\epsilon^4 \cdot (1+\epsilon)^k \leq 5/\epsilon^4 \cdot \lfloor (1+\epsilon)^k \rfloor = 5/\epsilon^4 \cdot q_k.$$

That is, the finish time of an edge is at most $5/\epsilon^4$ times its demand. Therefore the total finish time of the small edges is at most $5/\epsilon^4$ times the total demand, which is

$$\begin{aligned}
&\frac{5}{\epsilon^4}\sum_{i=0}^{N(v)-\lfloor 1/\epsilon^2\rfloor} n(v,i)q_i \le \frac{20}{\epsilon^7}\sum_{i=0}^{N(v)-\lfloor 1/\epsilon^2\rfloor}(1+\epsilon)^i \\
&\le \frac{21}{\epsilon^8}(1+\epsilon)^{N(v)-\lfloor 1/\epsilon^2\rfloor} \le \frac{22}{\epsilon^8}(1+\epsilon)^{N(v)-1/\epsilon^2} \\
&\le \frac{22}{\epsilon^8}\cdot 2^{-1/\epsilon}\cdot(1+\epsilon)^{N(v)} \le \frac{\epsilon^2}{2}\cdot\frac{1}{2}(1+\epsilon)^{N(v)} \\
&\le \frac{\epsilon^2}{2}\cdot q_{N(v)}.
\end{aligned}$$

(In the fourth inequality we use $(1+\epsilon)^{1/\epsilon} \ge 2$, in the fifth inequality it is assumed that ϵ is sufficiently small that $2^{1/\epsilon} \ge 44/\epsilon^{10}$ holds.) However, the small edges do not use the full color spectrum, only the colors in zone Z_r. By Lemma 7b, zone Z_r contains at least c colors up to $\lceil(1+\epsilon\ell)(1+\epsilon)/\epsilon\cdot c\rceil \le 2/\epsilon\cdot c$, thus every finish time in the calculation above should be multiplied by at most $2/\epsilon$. Therefore the sum of the small edges is

$$f_{\Psi_2}(R_v) \le 2/\epsilon\cdot\frac{\epsilon^2}{2}\cdot q_{N(v)} \le \epsilon q_{N(v)},$$

as claimed.

Phase 4: Shifting the Frequent Edges. Now we have a coloring Ψ_3 that is still not a proper coloring, but conflicts appear only between some frequent edges and their child edges. First we ensure that every frequent edge e uses only colors greater than $2x(e)/\epsilon$. After that, the conflicts are resolved using a set of so far unused colors, the colors in extra zones Z_5 and Z_6.

Let M_v be the set of frequent child edges of v, and let $\Lambda_v = \bigcup_{e\in M}\ \Psi_3(v)$ be the colors used by the frequent child edges of node v. We recolor the edges in M_v using only the colors in Λ_v. Let $e_1, e_2, \ldots, e_{|M\ |}$ be an ordering of the edges in M_v by increasing demand size. Recall that the algorithm in Theorem 9 assigned the colors to the split edges (leaf edges) in increasing order of demand size, thus it can be assumed that frequent edge e_1 uses the first $x(e_1)$ colors in Λ_v, edge e_2 uses the $x(e_2)$ colors after that, etc. Denote by $t(c) = |\{e \in M_v | f_{\Psi_3}(e) \ge c\}|$ the number of edges whose finish time is at least c, and denote by $t(c,i) = |\{e \in M_v | f_{\Psi_3}(e) \ge c,\ x(e) = q_i\}|$ the number of class i edges among them. Clearly, $t(c) = \sum_{i=0}^{\infty} t(c,i)$ holds. Moreover, it can be easily verified that the total finish time of the edges in M_v can be expressed as $f_{\Psi_3}(M_v) = \sum_{c=1}^{\infty} t(c)$.

The first step is to produce a coloring Ψ_4 where every frequent edge e has only $x(e)/(1+\epsilon)$ colors, but these colors are all greater than $2x(e)/\epsilon$. The demand function is split into two parts: $x(e) = x_1(e) + x_2(e)$, where $x_1(e)$ is $x(e)/(1+\epsilon)$ and $x_2(e)$ is $\epsilon x(e)/(1+\epsilon)$, rounding problems are ignored in this extended abstract.

This phase of the algorithm produces a coloring Ψ_4 of M_v that assigns only $x_1(e)$ colors to every edge $e \in M_v$, but satisfies the condition that it uses only the colors in Λ_v, and every edge e receives only colors greater than $2x(e)/\epsilon$. In the next phase we will extend this coloring using the colors in zones Z_3 and Z_4: every edge e will receive an additional $x_2(e)$ colors.

Coloring Ψ_4 is defined as follows. Consider the edges $e_1, \ldots, e_{|M|}$ in this order, and assign to e_k the first $x_1(e_k)$ colors in Λ_v greater than $2x(e_k)/\epsilon$ and not already assigned to an edge e_j ($j < k$). Notice the following property of Ψ_4: if $j < k$, then every color in $\Psi_4(e_j)$ is less than every color in $\Psi_4(e_k)$. This follows from $2x(e_j)/\epsilon \leq 2x(e_k)/\epsilon$: every color usable for e_k is also usable for e_j if $j < k$. Define $t'(c) = |\{e \in M_v | f_{\Psi_4}(e) \geq c\}|$ and $t'(c,i) = |\{e \in M_v | f_{\Psi_4}(e) \geq c,\ x(e) = q_i\}|$ as before, but now using the coloring Ψ_4. We claim that $t'(c,i) \leq (1+\epsilon)t(c,i)$ holds for every $c \geq 1, i \geq 0$. If this is true, then $t'(c) \leq (1+\epsilon)t(c)$ and $f_{\Psi_4}(M_v) \leq (1+\epsilon)f_{\Psi_3}(M_v)$ follow from $f_{\Psi_4}(M_v) = \sum_{c=1}^{\infty} t'(c)$. Summing this for every node v gives $f_{\Psi_4}(T) \leq (1+\epsilon)f_{\Psi_3}(T)$.

First we show that $t'(c,i) \leq t(c,i) + 3/\epsilon$. Denote by $\lambda_c = |\Lambda_v \cap [1, c-1]|$ the number of colors in Λ_v available below c. If every class i edge has finish time at least c in Ψ_3, then $t(c,i) = n(c,i) \geq t'(c,i)$ and we are ready. Therefore there is at least one class i edge that has finish time less than c. This implies that the frequent edges of class $0, 1, \ldots, i-1$ use only colors less than c. Denote by X the total demand of these edges (in the demand function $x(e)$), the class i frequent edges use at most $\lambda_c - X$ colors below c.

Now recall the way Ψ_4 was defined, and consider the step when every edge with class less than i is already colored. At this point at most X colors of Λ_c are used (possibly less, since Ψ_4 assigns only $x_1(e)$ colors to every edge e, and only colors above $2x(e)/\epsilon$). Therefore at least $\lambda_c - X$ colors are still unused in Λ_v below c. From these colors at least $\lambda_c - X - \lceil 2q_i/\epsilon \rceil$ of them are above $2q_i/\epsilon$. Thus Ψ_4 can color at least $(\lambda_c - X - \lceil 2q_i/\epsilon \rceil)/q_i \geq (\lambda_c - X)/q_i - 3/\epsilon$ edges of class i using only colors below c. However, Ψ_3 uses $\lambda_c - X$ colors below c for the class i edges, hence it can color at most $(\lambda_c - X)/q_i$ such edges below c, and $t'(c,i) \leq t(c,i) + 3/\epsilon$ follows.

We consider two cases. If $t(c,i) \geq 3/\epsilon^2$, then $t'(c,i) \leq t(c,i) + 3/\epsilon \leq (1+\epsilon)t(c,i)$, and we are ready. Let us assume therefore that $t(c,i) \leq 3/\epsilon^2$, it will turn out that in this case $t'(c,i) = 0$. There are $n(v,i) - t(c,i) \geq n(v,i) - 3/\epsilon^2$ class i edges that has finish time less than c in Ψ_3. Therefore, as in the previous paragraph, before Ψ_4 starts coloring the class i edges, there are at least $(n(v,i) - 3/\epsilon^2) \cdot q_i$ unused colors less than c in Λ_v. The total demand of the class i edges in $x_1(e)$ is at most $n(e,i)q_i/(1+\epsilon)$. The following calculation shows that the unused colors below c in Λ_v is sufficient to satisfy all these edges, thus Ψ_4 assigns to these edges only colors less than c. We have to skip the colors not greater than $2q_i/\epsilon$, these colors cannot be assigned to the edges of class i, which means that the number of usable colors is at least

$$\begin{aligned}(n(v,i) - 3/\epsilon^2) \cdot q_i - 2q_i/\epsilon &\geq (n(v,i) - \frac{1}{1+\epsilon} \cdot 4/\epsilon^2) \cdot q_i + 1 \\ &\geq (1 - \frac{\epsilon}{1+\epsilon}) n(v,i) q_i + 1 \geq n(e,i)q_i/(1+\epsilon),\end{aligned}$$

since $n(v,i) \geq 4/\epsilon^3$ by the definition of the frequent edges. Therefore Ψ_4 assigns to the class i edges only colors less than c, hence $t(c,i) = 0$, as claimed.

Phase 5: Full Demand for the Frequent Edges. The next step is to modify Ψ_4 such that every frequent edge receives $x(e)$ colors, not only $x_1(e)$. Coloring Ψ_5 is obtained from Ψ_4 by assigning to every frequent edge e an additional $x_2(e)$ colors from zone Z_3 or Z_4. More precisely, let v be a node with parity r, and let $e_1, \ldots, e_{|M|}$ be its frequent child edges, ordered in increasing demand size, as before. Assign to e_1 the first $x_2(e_1)$ colors from Z_{2+r}, to e_2 the first $x_2(e_2)$ colors from Z_{2+r} not used by e_1, etc. It is clear that no conflict arises with the assignment of these colors.

We claim that these additional colors increase the total finish time of the frequent edges at v by at most a factor of $(1 + (\ell + 1)\epsilon)$. Let $x_i^* = \sum_{j=1}^{i} x(e_j)$ be the total demand of the first i edges. The finish time of e_i in Ψ_4 is clearly at least x_i^*, since Ψ_4 colors every edge e_j with $j < i$ before e_i. On the other hand, by Lemma 7b, zone Z_{2+r} contains at least $\frac{\epsilon}{1+\epsilon} x_i^*$ colors not greater than $\lceil (1 + \epsilon\ell) x_i^* \rceil$. These colors are sufficient to satisfy the additional demand of the first i edges.

Phase 6: Resolving the Conflicts. Now we have a coloring Ψ_5 such that there are conflicts only between frequent edges and their child edges. Furthermore, if e is a frequent edge, then $\Psi_5(e)$ contains only colors greater than $2x(e)/\epsilon$ from the main zone. It is clear from the construction of Ψ_5 that only the colors in the main zone can conflict.

Let e be a frequent edge that conflicts with some of its children. Let the child edges of e have parity r. There are at most $x(e)$ colors that are used by both e and a child of e. We resolve this conflict by recoloring the child edges of e in such a way that they use the first at most $x(e)$ colors in zone Z_{4+r} instead of the colors in $\Psi_5(e)$. It is clear that if this operation is applied for every frequent edge e, then the resulting color Ψ_6 is a proper coloring.

Notice that if a child edge e' of e is recolored, then it has finish time at least $\lceil 2x(e)/\epsilon \rceil$, otherwise it does not conflict with e. On the other hand, by Lemma 7b, zone Z_{4+r} contains at least $x(e)$ colors up to $\lceil (1+\epsilon\ell) \cdot (1+\epsilon) x(e)/\epsilon \rceil \leq \lceil 2x(e)/\epsilon \rceil$, thus the recoloring does not add colors above that. Therefore the finish time of e' is not increased, since $f_{\Psi_5}(e') \geq \lceil 2x(e)/\epsilon \rceil$.

Analysis. If we follow how the sum of the coloring changes during the previous steps, it turns out that Ψ_6 is an ϵ_0-approximate solution to the instance (T, x_0).

The running time of the algorithm is dominated by the coloring of the low-degree components with the algorithm of Theorem 9. This phase requires $2^{O(16/\epsilon^{10} \cdot 1/\epsilon \cdot \log^2(4/\epsilon^6))} \cdot n) = 2^{O(1/\epsilon_0^{11} \log^2(1/\epsilon_0))} \cdot n$ time. The other parts of the algorithm can be done in time linear in the size of the input. Therefore the total running time is $2^{O(1/\epsilon_0^{11} \log^2(1/\epsilon_0))} \cdot n$, which completes the proof of Theorem 10. □

References

1. A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai. Sum multicoloring of graphs. *J. Algorithms*, 37(2):422–450, 2000.
2. E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers. *SIAM J. Comput.*, 14(3):744–780, 1985.
3. K. Giaro and M. Kubale. Edge-chromatic sum of trees and bounded cyclicity graphs. *Inform. Process. Lett.*, 75(1-2):65–69, 2000.
4. M. M. Halldórsson and G. Kortsarz. Tools for multicoloring with applications to planar graphs and partial k-trees. *J. Algorithms*, 42(2):334–366, 2002.
5. M. M. Halldórsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, and J. A. Telle. Multicoloring trees. *Inform. and Comput.*, 180(2):113–129, 2003.
6. J. A. Hoogeveen, S. L. van de Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Appl. Math.*, 55(3):259–272, 1994.
7. M. R. Salavatipour. On sum coloring of graphs. *Discrete Appl. Math.*, 127(3):477–488, 2003.

Scheduling to Minimize Average Completion Time Revisited: Deterministic On-Line Algorithms

Nicole Megow[1,⋆] and Andreas S. Schulz[2]

[1] Institut für Mathematik, Technische Universität Berlin, Sekr. MA 6-1, Strasse des 17. Juni 136, 10623 Berlin, Germany
nmegow@math.tu-berlin.de

[2] Sloan School of Management, Massachusetts Institute of Technology, Office E53-361, 77 Massachusetts Avenue, Cambridge, MA 02139-4307, USA
schulz@mit.edu

Abstract. We consider the scheduling problem of minimizing the average weighted completion time on identical parallel machines when jobs are arriving over time. For both the preemptive and the nonpreemptive setting, we show that straightforward extensions of Smith's ratio rule yield smaller competitive ratios compared to the previously best-known deterministic on-line algorithms, which are $(4+\varepsilon)$-competitive in either case. Our preemptive algorithm is 2-competitive, which actually meets the competitive ratio of the currently best randomized on-line algorithm for this scenario. Our nonpreemptive algorithm has a competitive ratio of 3.28. Both results are characterized by a surprisingly simple analysis; moreover, the preemptive algorithm also works in the less clairvoyant environment in which only the ratio of weight to processing time of a job becomes known at its release date, but neither its actual weight nor its processing time. In the corresponding nonpreemptive situation, every on-line algorithm has an unbounded competitive ratio.

1 Introduction

Model. We consider the problem of scheduling jobs arriving over time on-line on identical parallel machines to minimize the sum of weighted completion times. Each of the m machines can process only one of the n jobs at a time. Each job j of a given instance has a positive processing time $p_j > 0$ and a nonnegative weight $w_j \geq 0$. We learn about these job data only at the job's release date $r_j \geq 0$, which is not known in advance, either. If C_j denotes the completion time of job j in a feasible schedule, the corresponding objective function value is $\sum_{j=1}^{n} w_j C_j$. We consider both the preemptive and the nonpreemptive machine environments. In the preemptive setting, the processing of a job may

⋆ Supported by the DFG Research Center "Mathematics for key technologies" (FZT 86) in Berlin. Part of this research was performed while this author was visiting the Operations Research Center at the Massachusetts Institute of Technology.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 227–234, 2004.
© Springer-Verlag Berlin Heidelberg 2004

be suspended and resumed later on any machine at no extra cost. In contrast, once a job is started in the nonpreemptive mode, it must be processed on the same machine without any interruption until its completion. In scheduling notation [6], the corresponding off-line problems are denoted by $\mathrm{P} \mid r_j, \mathrm{pmtn} \mid \sum w_j C_j$ and $\mathrm{P} \mid r_j \mid \sum w_j C_j$, respectively. Already the analogous single-machine problems are NP-hard [9, 10].

However, instances of $1 \mid \mid \sum w_j C_j$ are optimally solved by Smith's Weighted Shortest Processing Time (WSPT) rule, which sequences jobs in nonincreasing order of their weight-to-processing-time ratios [17]. For convenience, we assume that the jobs are indexed in this order so that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$. Moreover, we say that a job with a smaller index has higher priority than one with a larger index.

The quality of on-line algorithms is typically assessed by their worst-case performance, expressed as the competitive ratio [16]. A ρ-competitive algorithm provides for any instance a solution with an objective function value of at most ρ times the value of an optimal off-line solution.

Main Results. We show in Section 2 that a natural extension of the WSPT rule to preemptive scheduling on identical parallel machines with release dates is 2-competitive, and this bound is tight. The idea is to interrupt currently active jobs of lower priority whenever new high-priority jobs arrive and not enough machines are available to accommodate the arrivals.

When preemption is not allowed, a straightforward extension of this scheme is to start the currently available job of highest priority whenever a machine becomes idle. However, this rule does not directly lead to a bounded competitive ratio. In fact, consider a single-machine instance in which a job of high priority is released right after the start of a long lower-priority job. Therefore, we first modify the release date of each job such that it is equal to a certain fraction of its processing time, if necessary. If we now start a long job j and a high-priority job becomes available shortly thereafter, the ill-timed choice of starting j can be accounted for by the fact that the high-priority job has a release date or processing time at least as large as a fraction of p_j. Therefore, its delay is bounded by its own contribution to the objective function in any feasible schedule. We consider a family of alike algorithms in Section 3 and show that the best one is 3.28-competitive. In this case, we cannot show that our analysis is tight, but the remaining gap is at most 0.5.

Related Work. Lu, Sitters and Stougie [11] introduced a related class of 2-competitive algorithms, which use similar waiting strategies for the on-line variant of the single-machine problem $1 \mid r_j \mid \sum C_j$. In fact, the idea of boosting release dates was used before by Hoogeveen and Vestjens [8] and Stougie (cited in [18]), who delayed the release date of each job j until time $\max\{r_j, p_j\}$ and $r_j + p_j$, respectively. Anderson and Potts [2] extended both, Hoogeveen and Vestjen's algorithm and its competitive ratio of 2, to the setting of arbitrary nonnegative job weights. These results are best possible since Hoogeveen and

Vestjens also proved that no nonpreemptive deterministic on-line algorithm can achieve a competitive ratio better than 2.

Phillips, Stein and Wein [12] presented another on-line algorithm for $1 \mid r_j \mid \sum C_j$, which converts a preemptive schedule into a nonpreemptive one of objective function value at most twice that of the preemptive schedule. Since Schrage's Shortest Remaining Processing Time (SRPT) rule [13] works on-line and produces an optimal preemptive schedule for the single-machine problem, it follows that Phillips, Stein and Wein's algorithm has competitive ratio 2 as well. The conversion factor is $3 - 1/m$ if applied to identical parallel machines, but the corresponding preemptive problem is NP-hard in this case. However, Chekuri, Motwani, Natarajan and Stein [3] noted that by sequencing jobs nonpreemptively in the order of their completion times in the optimal preemptive schedule on a single machine of speed m times as fast as that of any one of the m parallel machines, one obtains a $(3 - 1/m)$-competitive algorithm for the on-line version of $\mathrm{P} \mid r_j \mid \sum C_j$. For the same problem, Lu, Sitters and Stougie [11] gave a 2α-competitive algorithm, where α is the competitive ratio of the direct extension of the SRPT rule to identical parallel machines. Phillips, Stein and Wein [12] showed that this rule is 2-competitive, but a smaller value of α has not been ruled out.

In any case, the hitherto best known deterministic on-line result for the corresponding scheduling problems with arbitrary job weights, $\mathrm{P} \mid r_j, \mathrm{pmtn} \mid \sum w_j C_j$ and $\mathrm{P} \mid r_j \mid \sum w_j C_j$ was a $(4+\varepsilon)$-competitive algorithm by Hall, Schulz, Shmoys and Wein [7], which was given as part of a more general on-line framework. For $1 \mid r_j, \mathrm{pmtn} \mid \sum w_j C_j$, Goemans, Wein and Williamson (cited as personal communication in [14]) noted that the preemptive version of the WSPT rule is 2-competitive; it schedules at any point in time the highest-priority job, possibly preempting jobs of lower priority. (A proof of this result is given in [14].) Our preemptive parallel-machine algorithm is the direct extension of this variation of Smith's rule. Schulz and Skutella [14] and Goemans, Queyranne, Schulz, Skutella and Wang [5] give comprehensive reviews of the development of on-line algorithms for the preemptive and nonpreemptive single-machine problems, respectively; Hall, Schulz, Shmoys and Wein [7] do the same for the parallel machine counterparts.

On the side of negative results, Vestjens [18] proved a universal lower bound of 1.309 for the competitive ratio of any deterministic on-line algorithm for $\mathrm{P} \mid r_j \mid \sum C_j$. In the preemptive case, the currently known lower bound is just $22/21$, also given by Vestjens.

Let us eventually mention that the currently best randomized on-line algorithms for the two problems considered here have (expected) competitive ratio 2; see [15]. Moreover, the off-line versions of these problems are well understood; in fact, both problems have a polynomial-time approximation scheme [1].

2 Preemptive Parallel Machine Scheduling

We consider the following extension of the single-machine preemptive WSPT rule to identical parallel machines.

Algorithm P-WSPT
At any point in time, schedule the m jobs with the highest priorities among the available, not yet completed jobs (or fewer if less than m incomplete jobs are available). Interrupt the processing of currently active jobs, if necessary.

The algorithm works on-line since the decision about which job to run at any given point in time t is just based on the set of available jobs at time t. In fact, it only depends on the priorities of the available jobs. In particular, Algorithm P-WSPT also operates in an environment in which actual job weights and processing times may not become available before the completion of the jobs, as long as the jobs' priorities are known at their respective release dates.

Theorem 1. *The Algorithm* P-WSPT *produces a solution of objective function value at most twice the optimal value for the off-line problem* $P|r_j, pmtn| \sum w_j C_j$.

Proof. Consider the time interval $(r_j, C_j]$ of an arbitrary but fixed job j. We partition this interval into two disjunctive sets of subintervals: $I(j)$ contains the subintervals in which job j is being processed, and $\bar{I}(j)$ denotes the set of remaining subintervals, in which other jobs than j are being processed. Note that no machine can be idle during the subintervals belonging to $\bar{I}(j)$. Since the algorithm processes job j after its release date r_j whenever a machine is idle, we obtain

$$C_j \leq r_j + |I(j)| + |\bar{I}(j)| \ ,$$

where $|\cdot|$ denotes the sum of the lengths of the subintervals in the corresponding set.

The overall length of $I(j)$ is clearly p_j. Only jobs with a higher ratio of weight to processing time than j can be processed during the intervals of the set $\bar{I}(j)$, because the algorithm gives priority to j before scheduling jobs with lower ratio. In the worst case, that is when $|\bar{I}(j)|$ is maximal, all jobs with higher priority than j are being processed in the subintervals of this set. Then $|\bar{I}(j)| = (\sum_{k<j} p_k)/m$, and we can bound the value of the P-WSPT schedule as follows:

$$\sum_j w_j C_j \ \leq \ \sum_j w_j (r_j + p_j) + \sum_j w_j \sum_{k<j} \frac{p_k}{m} \ .$$

Since the completion time C_j of a job j is always at least as large as its release date plus its processing time, $\sum_j w_j (r_j + p_j)$ is obviously a lower bound on the value of an optimal schedule. Moreover, $\sum_j w_j \sum_{k \leq j} p_k/m$ is the objective function value of an optimal solution to the corresponding instance of the relaxed problem $1\,||\sum w_j C_j$ on a single machine with speed m times the speed of any of the identical parallel machines. As this problem is indeed a relaxation of the scheduling problem considered here, we can conclude that the P-WSPT algorithm is 2-competitive. □

A family of instances provided by Schulz and Skutella [14] shows that this result cannot be improved. In fact, for $m = 1$, P-WSPT coincides with the preemptive single-machine algorithm studied in their paper. Taking m copies of Schulz and Skutella's instance yields the following result.

Lemma 2. *The competitive ratio of the Algorithm* P-WSPT *is not better than* 2 *for the on-line problem* $P\,|\,r_j, pmtn\,|\,\sum w_jC_j$*, for any given number of machines.*

Proof. We include a proof for the sake of completeness. We consider an instance that is slightly different from the one given in [14]. It consists of m copies of $n+1$ jobs with $w_j = 1$, $p_j = n - j/n$ and $r_j = jn - j(j+1)/(2n)$ for all $0 \le j \le n$. Algorithm P-WSPT preempts any job when it has left just $1/n$ units of processing time and finishes it only after all jobs with a larger release date have been completed. The value of this schedule is $m \cdot (\sum_{j=0}^{n}(r_n + p_n + j/n))$.

An optimal off-line algorithm does not preempt any job and yields a schedule of value $m \cdot (\sum_{j=0}^{n}(r_j + p_j + j/n))$. A simple calculation shows that the ratio of the values of the P-WSPT schedule and the optimal schedule goes to 2 when n goes to infinity. □

Of course, Theorem 1 subsumes the scheduling problem $\mathrm{P}\,|\,r_j, \mathrm{pmtn}\,|\,\sum C_j$ as a special case. Thus, this extension of the 2-competitive single-machine Shortest Processing Time (SPT) rule to the parallel machine setting has the same competitive ratio as the analogous extension of Schrage's optimal single-machine SRPT rule [12].

3 Nonpreemptive Parallel Machine Scheduling

Every reasonable on-line algorithm for nonpreemptive scheduling has to make use of some kind of waiting strategy. We refer the reader to [18, Chapter 2] and [11] for comprehensive discussions of related techniques for the single machine. Here, we extend the idea of delaying release dates to the parallel machine problem.

> **Algorithm** Shifted WSPT
> Modify the release date of every job j to r'_j, where r'_j is some value between $\max\{r_j, \alpha\, p_j\}$ and $r_j + \alpha\, p_j$, for some $\alpha \in (0, 1]$. Whenever a machine becomes idle, choose among the available jobs a job j with highest priority and schedule it on the idle machine.

Note that this is indeed an on-line algorithm; we will later choose α so that we minimize the corresponding competitive ratio. Moreover, for $m = 1$ and $\alpha = 1$, Algorithm Shifted WSPT is identical to the algorithm proposed in [11] for $1\,|\,r_j\,|\,\sum C_j$.

In the analysis of the Shifted WSPT algorithm, we make use of the following lower bound on the optimal value of the relaxed problem with trivial release dates, which is due to Eastman, Even and Isaacs [4].

Lemma 3. *The value of an optimal schedule for an instance of the scheduling problem* $P\,||\,\sum w_jC_j$ *is bounded from below by*

$$\sum_{j=1}^{n} w_j \sum_{k \le j} \frac{p_k}{m} + \frac{m-1}{2m} \sum_{j=1}^{n} w_j p_j.$$

Let us now analyze the performance of the SHIFTED WSPT algorithm.

Theorem 4. *The Algorithm* SHIFTED WSPT *has a competitive ratio of less than* $2 + \max\{1/\alpha, \alpha + \frac{m-1}{2m}\}$ *for the on-line problem* $P|r_j| \sum w_j C_j$.

Proof. The algorithm schedules a job j at time r'_j if a machine is idle and j is the job with the highest ratio of weight to processing time among all available jobs; otherwise, j has to wait for some time. The waiting time for j after r'_j is caused by two types of jobs: jobs with lower priority but which started before time r'_j, and jobs with higher priority. Let $L(j)$ denote the set of lower-priority jobs $\ell > j$ with $r'_\ell < r'_j$ and $C_\ell > r'_j$, and let $H(j)$ denote the set of higher-priority jobs $h < j$ that postpone the start of j. Then,

$$C_j \le r'_j + W(L(j)) + W(H(j)) + p_j \ , \tag{1}$$

where $W(L(j))$ and $W(H(j))$ denote the waiting time for j caused by jobs in the sets $H(j)$ and $L(j)$, respectively. Note that the algorithm does not insert idle time on any machine in the time interval between r'_j and the start of job j. Let us analyze the maximal waiting time that can be caused by jobs in the set $L(j)$ and in the set $H(j)$, separately.

1. *Jobs in* $L(j)$. Each machine has at most one job $\ell \in L(j)$. By definition, every one of these jobs satisfies $\alpha\, p_\ell \le r'_\ell < r'_j$. Thus, $p_\ell < r'_j/\alpha$.
2. *Jobs in* $H(j)$. The maximal waiting time for job j arises if all jobs $h < j$ are processed after the completion of jobs in the set $L(j)$ and before the start of job j. If all jobs in $H(j)$ are simultaneously processed on m machines, then at least one machine finishes processing jobs in $H(j)$ after a time period of length at most $\sum_{h\in H(j)} p_h/m$. Hence, the maximal waiting time for job j caused by jobs in $H(j)$ is $\sum_{h<j} p_h/m$.

Inequality (1) and the observations on the sets causing waiting time imply:

$$\begin{aligned} C_j &< (1 + \frac{1}{\alpha})\ r'_j + \sum_{h<j} \frac{p_h}{m} + p_j \\ &\le (1 + \frac{1}{\alpha})(r_j + \alpha p_j) + \frac{m-1}{2m} p_j + \sum_{h\le j} \frac{p_h}{m} + \frac{m-1}{2m} p_j \\ &= (\ \frac{r_j}{\alpha} + (\alpha + \frac{m-1}{2m})\ p_j\ ) + (r_j + p_j) + \sum_{h\le j} \frac{p_h}{m} + \frac{m-1}{2m}\ p_j\ . \end{aligned}$$

Thus, Algorithm SHIFTED WSPT generates a schedule of value

$$\sum_j w_j C_j < (1 + \max\{\tfrac{1}{\alpha}, \alpha + \tfrac{m-1}{2m}\}) \sum_j w_j (r_j + p_j) + \sum_j w_j (\sum_{h\le j} \tfrac{p}{m} + \tfrac{m-1}{2m}\ p_j).$$

The proof is completed by applying two lower bounds on the optimal value: first, the trivial lower bound $\sum_j w_j(r_j + p_j)$, and second, the lower bound presented in Lemma 3. □

A simple calculation shows that the minimum of $\max\{\frac{1}{\alpha}, \alpha+\frac{m-1}{2m}\}$ is attained at $\alpha = (1-m+\sqrt{16m^2+(m-1)^2})/(4m) =: \alpha_m$. In particular, $\alpha_1 = 1$.

Corollary 5. *The Algorithm* SHIFTED WSPT *with* $\alpha = \alpha_m$ *is* $(2+1/\alpha_m)$*-competitive. The value of this increasing function of* m *is* 3 *for the single-machine case and has its limit at* $(9+\sqrt{17})/4 \approx 3.28$ *for* $m \to \infty$.

Lemma 6. *Algorithm* SHIFTED WSPT *cannot achieve a better competitive ratio than* $\max\{2+\alpha, 1+\frac{1}{\alpha}\} \geq 2+\frac{\sqrt{5}-1}{2}$ *for* $\alpha \in (0,1]$*, on any number of machines.*

Proof. We give two instances from which the lower bound follows. Consider $2m$ jobs released at time 0; half of the jobs are of type I and have unit processing times and weights ε, whereas the other half of the jobs, type II, have processing time $1+\varepsilon$ and unit weight. The algorithm modifies the release dates and schedules jobs of type I at time $t = \alpha$ first, one on each machine, followed by jobs of type II. The value of the schedule is $m(\alpha+1)\varepsilon + m(\alpha+2+\varepsilon)$. In the optimal schedule, jobs of type II start processing first, at time $t = 0$, such that the value of the schedule is $m(1+\varepsilon) + m(2+\varepsilon)\varepsilon$. For $\varepsilon \to 0$, the ratio between the value of the SHIFTED WSPT schedule and the optimal schedule goes to $2+\alpha$.

The second instance consists again of $2m$ jobs: half of the jobs are of type I and have release dates $1+\varepsilon$, processing times ε and weights $1/m$, whereas the other half of the jobs, type II, are released at time 0 and have processing time $1/\alpha$ and zero weight. SHIFTED WSPT starts scheduling jobs at time 1 and obtains a solution with a value of at least $1+1/\alpha+\varepsilon$. The value of the optimal schedule is $1+2\varepsilon$. □

For the special choice $\alpha = \alpha_m$, the first lower bound is tighter and it follows more concretely:

Corollary 7. *The Algorithm* SHIFTED WSPT *with* $\alpha = \alpha_m$ *is not better than* $(1+7m+\sqrt{16m^2+(m-1)^2})/(4m)$ *for instances with* m *machines. This means that our performance analysis has a gap of at most* $(m-1)/2m < 0.5$*, and it is tight for the single-machine problem.*

References

1. F.N. Afrati, E. Bampis, C. Chekuri, D.R. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, New York, NY, pages 32–43, 1999.
2. E.J. Anderson and C.N. Potts. On-line scheduling of a single machine to minimize total weighted completion time. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, San Francisco, CA, pages 548–557, 2002.
3. C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31:146–166, 2001.

4. W.L. Eastman, S. Even, and I.M. Isaacs. Bounds for the optimal scheduling of n jobs on m processors. *Management Science*, 11:268–279, 1964.
5. M.X. Goemans, M. Queyranne, A.S. Schulz, M. Skutella, and Y. Wang. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics*, 15:165–192, 2002.
6. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
7. L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.
8. J.A. Hoogeveen and A.P.A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In W.H. Cunningham, S.T. McCormick, and M. Queyranne, editors, *Proceedings of the Fifth Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 1084 of *Lecture Notes in Computer Science*, pages 404–414, Springer, Berlin, 1996.
9. J. Labetoulle, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In W.R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 245–261, Academic Press, New York, 1984.
10. J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.
11. X. Lu, R.A. Sitters, and L. Stougie. A class of on-line scheduling algorithms to minimize total completion time. *Operations Research Letters*, 31:232–236, 2003.
12. C.A. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223, 1998.
13. L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:687–690, 1968.
14. A.S. Schulz and M. Skutella. The power of α-points in preemptive single machine scheduling. *Journal of Scheduling*, 5:121–133, 2002.
15. A.S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15:450–469, 2002.
16. D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.
17. W.E. Smith. Various optimizers for single-stage production. *Naval Research and Logistics Quarterly*, 3:59–66, 1956.
18. A.P.A. Vestjens. *On-line Machine Scheduling*. PhD thesis, Eindhoven University of Technology, Netherlands, 1997.

On-Line Extensible Bin Packing with Unequal Bin Sizes*

Deshi Ye[1] and Guochuan Zhang[1,2,**]

[1] Department of Mathematics,
Zhejiang University, Hangzhou 310027, China
[2] Institut für Informatik und Praktische Mathematik,
Christian-Albrechts-Universität zu Kiel,
Olshausenstrasse 40, 24098 Kiel, Germany
gzh@informatik.uni-kiel.de

Abstract. In the extensible bin packing problem we are asked to pack a set of items into a given number of bins, each with an original size. However, the original bin sizes can be extended if necessary. The goal is to minimize the total size of the bins. We consider the problem with unequal (original) bin sizes and present the tight bound of a list scheduling algorithm for each collection of original bin sizes and each number of bins. We further give better on-line algorithms for the two-bin case and the three-bin case. Interestingly, it is shown that the on-line algorithms have better competitive ratios for unequal bins than for equal bins. Some variants of the problem are also discussed.

1 Introduction

We consider the following on-line extensible bin packing problem: there are m bins $B_1, B_2, \ldots, B_m$ with original bin sizes $b_1, b_2, \ldots, b_m$. The original bin sizes can be extended if needed. The (final) size of a bin is defined to be the larger one between the original bin size and the total size of the items assigned to it. Items $\{a_1, a_2, \ldots, a_n\}$ arrive over list, which are not known in advance. When an item a_i arrives it must immediately and irrevocably be assigned to one of the bins and the next item a_{i+1} becomes known only after a_i has been assigned. The size of item a_i is p_i. The goal is to assign all items to the bins such that the total size of the bins is minimized.

This problem arises in a wide variety of contexts. It has many applications in bin packing, storage allocation and scheduling problems. Consider, for instance, a set of workers is given with regular working times. In case of needed, the worker can do some overtime works. The problem is to assign duties to the workers in such a way that the total work (regular working times plus overtime works) is minimized.

* Supported by EU-Project CRESCCO (IST-2001-33135) and NSFC (10231060).
** Corresponding author.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 235–247, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Competitive Ratios. An instance of the problem consists of a list of items and m bins with original sizes $b_1, b_2, \ldots, b_m$. Let $\mathcal{B}$ be the collection of bin sizes, i.e., $\mathcal{B} = \{b_1, b_2, \cdots, b_m\}$. For any instance L, let $A_L(m, \mathcal{B})$ and $OPT_L(m, \mathcal{B})$ be the total size of the bins used by an on-line algorithm A and the total size of the bins used by an optimal off-line algorithm, respectively. Then the competitive ratio of algorithm A is

$$R_A(m, \mathcal{B}) = \sup_L A_L(m, \mathcal{B})/OPT_L(m, \mathcal{B}).$$

If the bin sizes are identical, i.e., $b_1 = b_2 = \cdots = b_m$, the competitive ratio is denoted as $R_A(m)$. Further we define the overall competitive ratios as follows.

$$R_A(m, \cdot) = \sup_{\mathcal{B}} R_A(m, \mathcal{B}), \quad R_A = \sup_{m, \mathcal{B}} R_A(m, \mathcal{B}).$$

Known Results. The extensible bin packing problem with unequal bin sizes was first studied by Dell'Olmo and Speranza [5]. They considered both the off-line case and the on-line case under the assumption that the maximum item size is not larger than the minimum bin size. They showed an upper bound of $4 - 2\sqrt{2}$ for an LPT algorithm in the off-line case and an upper bound of $5/4$ for a list scheduling algorithm LS in the on-line case. The bound $5/4$ is tight in terms of the overall competitive ratio, i.e., $R_{LS} = 5/4$.

The special case that all bin sizes are equal to one has been extensively studied. It was proved that the off-line version of the problem is NP-hard in the strong sense [4]. Actually, it can be easily reduced from the 3-partition problem [7]. Dell'Olmo et al. [4] proved that the worst-case ratio of the LPT algorithm is $13/12$. If the number of the bins is fixed, it follows from the results of [11] that there exists a fully polynomial time approximation scheme. If m is not fixed, a polynomial time approximation scheme was provided in [1]. Recently, Coffman and Lueker [3] presented a fully polynomial time asymptotic approximation scheme. They also proved that the problem does not admit a fully polynomial time approximation scheme unless P = NP, when m is not fixed. The on-line version was studied by Speranza and Tuza [10]. They showed that the overall competitive ratio of a list scheduling heuristic is $5/4$. Then a class of heuristics H_x were presented which depend on a parameter x, $0 < x < 1$, and tend to load partially loaded bins until a limit total load of $1 + x$ is reached on the bin. For any number of bins, the algorithm has an overall competitive ratio bounded above by 1.228. For the lower bound of the on-line problem, they showed that no heuristic can have a competitive ratio smaller than $7/6$ by presenting an instance for the case that $m = 2$. Ye and Zhang [12] considered on-line scheduling on a small number m of bins. They proved lower bounds for $m = 3, 4$ and gave the competitive ratios $R_H\ (m)$ for $m = 2, 3$ and 4. For $m = 2$ the algorithm is best possible. An improved algorithm for $m = 3$ was also presented.

A similar problem, called *on-line bin stretching*, was introduced by Azar and Regev [2]. A set of bins of fixed sizes is given. Items arrive on-line while it is known that there exists a valid packing of all items into the given bins. One is asked to pack all items into the bins, which can be overloaded. The goal function

is the stretch factor, which is the maximum ratio for any bin between the size to which any bin was stretched (the sum of all items assigned to it) and its original size. Azar and Regev [2] studied the problem of identical bins, i.e. all original bins are of size 1. Epstein [6] considered the two-bin case with unequal bin sizes.

Our Results. In this paper, firstly we assume, as in [5], that the largest item size is at most the smallest bin size, i.e.,

$$\max_{i=1}^{n} p_i \leq \min_{j=1}^{m} b_j. \tag{1}$$

We analyze the LS algorithm more carefully and prove the competitive ratios for each m and each collection $\mathcal{B}$ of bin sizes. The ratios differ for an even number of bins and an odd number of bins. It also shows that the competitive ratio of LS for the equal bins is exactly $5/4$ when m is even and $5/4 - 1/(4m^2)$ when m is odd. We then present an improved on-line algorithm for $m = 2$ and $m = 3$. Finally we discuss the problem without the assumption (1). A lower bound $6/5$ for the overall competitive ratio is presented. We prove the competitive ratio of LS for the two-machine case and present an improved on-line algorithm.

Notations and Organization of the Paper. Consider any algorithm A. During the execution of algorithm A, if a bin B_j has a load (the total size of items assigned to it) is over its original size b_j, B_j is called *heavy*; otherwise, it is called *light*. The difference between the load and the size b_j is *idle space* if B_j is light or *excess* if B_j is heavy.

Consider any instance L where $\sum_{i=1}^{n} p_i < \sum_{j=1}^{m} b_j$. In an optimal packing there must be some bins with idle space. Construct a new instance L' by adding new items such that no bins have idle space in the optimal packing without changing the optimal value. Note that $A_{L'}(m, \mathcal{B}) \geq A_L(m, \mathcal{B})$ and $OPT_{L'}(m, \mathcal{B}) = OPT_L(m, \mathcal{B})$. Therefore,

$$A_L(m, \mathcal{B})/OPT_L(m, \mathcal{B}) \leq A_{L'}(m, \mathcal{B})/OPT_{L'}(m, \mathcal{B}).$$

In instance L', the total size of items is at least the total original size of bins. It implies that we only need to consider the instances in which the total size of items is at least the total original size of bins to prove an upper bound. Throughout the paper in proving the competitive ratios we always assume

$$\sum_{i=1}^{n} p_i \geq \sum_{j=1}^{m} b_j. \tag{2}$$

The remainder of the paper is organized as follows. Section 2 gives the tight bound for a list scheduling algorithm. In Section 3, we present an on-line algorithm for $m = 2$ and $m = 3$. We discuss the problem without the item size assumption (1) in Section 4.

2 Tight Bound for a List Scheduling Algorithm

List Scheduling was introduced by Graham [8] for parallel machine scheduling. The algorithm always schedules the current job to the machine with minimum load at this time. Applying list scheduling to extensible bin packing with unequal bin sizes may result in different versions. We first consider a list scheduling algorithm, denoted by LS, which always assigns the incoming item to the bin of largest idle space. Dell'Olmo and Speranza [5] proved that $R_{LS} = 5/4$. In this section we figure out $R_{LS}(m, \mathcal{B})$. Then we will show that a different version of list scheduling, which always assigns the incoming item to the bin with the least load (the total size of items in a bin), is not as good as LS in terms of the overall competitive ratio.

Let b_{min} be the smallest bin sizes in the collection $\mathcal{B}$ and let p_{max} be the largest item size. Under the assumption (1), we have $p_{max} \leq b_{min}$. Consider the packing given by algorithm LS. For each bin B_j, let l_j be the total size of the items it accommodates after the schedule is over. If B_j is light, let s_j be its idle space, i.e., $s_j = b_j - l_j$. If B_j is heavy, let r_j be its idle space immediately before it becomes heavy, and e_j be its excess, i.e., $e_j = l_j - b_j$. We re-index the bins so that the first k bins are heavy.

In proving an upper bound for algorithm LS we can assume, without loss of generality, that a bin accepts no more items once it becomes heavy. Otherwise, all bins are heavy and the packing is optimal.

Lemma 2.1. *The competitive ratio of the list scheduling algorithm*

$$R_{LS}(m, \mathcal{B}) \geq \begin{cases} 1 + \frac{mb}{4\sum_{\ell=1} b_\ell}, & \text{if } m \text{ is even,} \\ 1 + \frac{(m^2-1)b}{4m\sum_{\ell=1} b_\ell}, & \text{if } m \text{ is odd.} \end{cases}$$

Proof. Consider the following instances. If m is even, a large number of small enough items are coming first. The total size of these items is $\sum_{j=1}^{m} b_j - mb_{min}/2$ such that after assigning these items by LS, each bin has idle space exactly $b_{min}/2$. Then $m/2$ items with size of b_{min} are coming. Clearly, the optimal value is $\sum_{j=1}^{m} b_j$, while LS generates a total size $\sum_{j=1}^{m} b_j + mb_{min}/4$. Thus $R_{LS}(m, \mathcal{B}) \geq 1 + \frac{mb}{4\sum_{\ell=1} b_\ell}$.

If m is odd, the instance is slightly different. A large number of small enough items are coming first. The total size of them is $\sum_{j=1}^{m} b_j - (m-1)b_{min}/2$, so that after assigning these items each bin has idle space exactly $\frac{m-1}{2m} b_{min}$. Then $(m-1)/2$ items with length b_{min} are coming. Clearly, the optimal total bin size is $\sum_{j=1}^{m} b_j$, while LS generates a total bin size of $\sum_{j=1}^{m} b_j + (m^2-1)b_{min}/(4m)$. Thus $R_{LS}(m, \mathcal{B}) \geq 1 + \frac{(m^2-1)b}{4m\sum_{\ell=1} b_\ell}$. The lemma is proved. □

Theorem 2.2. *The competitive ratio of the list scheduling algorithm*

$$R_{LS}(m, \mathcal{B}) = \begin{cases} 1 + \frac{mb}{4\sum_{\ell=1} b_\ell}, & \text{if } m \text{ is even,} \\ 1 + \frac{(m^2-1)b}{4m\sum_{\ell=1} b_\ell}, & \text{if } m \text{ is odd.} \end{cases}$$

Proof. We only need to prove that

$$R_{LS}(m,\mathcal{B}) \le \begin{cases} 1+\frac{mb}{4\sum_{=1} b}, & \text{if } m \text{ is even,} \\ 1+\frac{(m^2-1)b}{4m\sum_{=1} b}, & \text{if } m \text{ is odd.} \end{cases}$$

Consider any instance $L(m,\mathcal{B})$ satisfying (2). By the algorithm LS, we know that

$$\frac{\sum_{j=1}^{k} r_j}{k} \ge \min_{j=1,\cdots,k} r_j \ge \max_{j=k+1,\cdots,m} s_j \ge \frac{\sum_{j=k+1}^{m} s_j}{m-k}.$$

Then

$$(m-k)\sum_{j=1}^{k} r_j \ge k \sum_{j=k+1}^{m} s_j \tag{3}$$

From (2), we have

$$\sum_{i=1}^{k} e_i \ge \sum_{j=k+1}^{m} s_j. \tag{4}$$

We add $(m-k)\sum_{j=1}^{k} e_j$ to both sides of the inequality (3) and from (4), we get

$$\begin{aligned} (m-k)\sum_{j=1}^{k}(r_j+e_j) &\ge k\sum_{j=k+1}^{m} s_j + (m-k)\sum_{j=1}^{k} e_j \\ &\ge k\sum_{j=k+1}^{m} s_j + (m-k)\sum_{j=k+1}^{m} s_j \\ &= m\sum_{j=k+1}^{m} s_j. \end{aligned}$$

It means that $\sum_{j=k+1}^{m} s_j \le \frac{(m-k)}{m}\sum_{j=1}^{k}(r_j+e_j)$. Note that the competitive ratio

$$R_{LS}(m,\mathcal{B}) \le (\sum_{i=1}^{n} p_i + \sum_{j=k+1}^{m} s_j)/\sum_{i=1}^{n} p_i.$$

Then

$$R_{LS}(m,\mathcal{B}) \le 1+\frac{(m-k)\sum_{j=1}^{k}(r_j+e_j)}{m\sum_{i=1}^{n} p_i} \le 1+\frac{(m-k)\sum_{j=1}^{k}(r_j+e_j)}{m\sum_{j=1}^{m} b_j}.$$

Since $r_j + e_j \le p_{\max} \le b_{min}$, we obtain

$$R_{LS}(m,\mathcal{B}) \le 1+\frac{(m-k)kb_{min}}{m\sum_{j=1}^{m} b_j}.$$

If m is even, then $m^2 - 4mk + 4k^2 = (m-2k)^2 \geq 0$ and $(m-k)k/m^2 \leq 1/4$. It follows

$$R_{LS}(m, \mathcal{B}) \leq 1 + \frac{mb_{min}}{4\sum_{j=1}^{m} b_j}.$$

If m is odd, then $m^2 - 4mk + 4k^2 = (m-2k)^2 \geq 1$ and $(m-k)k \leq \frac{1}{4}(m^2-1)$. We have

$$R_{LS}(m, \mathcal{B}) \leq 1 + \frac{(m^2-1)b_{min}}{4m\sum_{j=1}^{m} b_j}.$$

□

Corollary 2.3.

$$R_{LS}(m, \mathcal{B}) \leq R_{LS}(m) = \begin{cases} \frac{5}{4}, & \text{if } m \text{ is even,} \\ \frac{5}{4} - \frac{1}{4m^2}, & \text{if } m \text{ is odd.} \end{cases}$$

Moreover, $R_{LS} = 5/4$. □

Note that $R_{LS}(m, \mathcal{B}) < R_{LS}(m)$ if $m \geq 2$ and $b_i \neq b_j$ for some i, j. It means that the competitive ratio becomes smaller if the bin sizes are unequal.

Before closing this section, we consider a different version of list scheduling: assign items to the bin of least load. Denote the algorithm as LS'. Consider the following simple instance L for two bins. Let $b_1 = 2b_2$ and let $\varepsilon > 0$ be a sufficiently small number. The first two items have size of $b_2 - \varepsilon$, which are followed by an item with size of $\varepsilon/2$. By algorithm LS', after assigning the first two items, each bin has a load of $b_2 - \varepsilon$. If the 3rd item goes to the first bin B_1, an item (the last one) with size of b_2 comes, which is assigned to B_2. If the 3rd item goes to the second bin B_2, then the 4th item has a size of ε and the 5th item (the last one) has size of b_2. The 4th is assigned to B_1 while the 5th is assigned to B_2. In both cases, $LS'_L(2, \mathcal{B}) = b_1 + b_2 + b_2 - \varepsilon$. In an optimal packing, we can assign the last item to B_2 and the others to B_1. Then $OPT_L(2, \mathcal{B}) = b_1 + b_2$, which shows that $R_{LS'}(2, \mathcal{B}) \to 4/3$ as ε tends to zero. It implies that $R_{LS'} \geq 4/3$.

3 The Two- and Three-Bin Cases

Assume that $b_1 \geq b_2 \geq \cdots \geq b_m \geq p_{max}$. In this section we consider the cases that $m = 2$ and $m = 3$.

Algorithm $A_m(\alpha)$: Assign the incoming item a_i to the lowest indexed bin, if the total excess of the bins is not greater than α; otherwise assign a_i to the bin of largest idle space.

In this algorithm α is a user-specified parameter. Choosing different parameters results in different algorithms.

Lemma 3.1. *For any parameter $\alpha > 0$, the competitive ratio of algorithm $A_2(\alpha)$ is at least $1 + b_2/(3(b_1 + b_2))$.*

Proof. Let N be a sufficiently large integer. If $\alpha < b_2/3$, consider an instance L as follows. The first $\lfloor (b_1 - 2b_2/3)N \rfloor$ items are small items, each with size $1/N$, which are followed by items a_1, a_2, a_3 with size of $b_2/3, 2b_2/3$ and $2b_2/3$, respectively. Clearly, $A_2(\alpha)$ assigns all the small items together with a_1 to bin B_1, and a_2 to bin B_2. Thus no matter where a_3 is assigned, $A_2(\alpha)_L(2, \mathcal{B}) \geq b_1 + b_2 + b_2/3 - 1/N$. For an optimal solution, we can assign a_1 and a_2 to B_2, and the remaining items to B_1. Thus $OPT_L(2, \mathcal{B}) = b_1 + b_2$, which follows that $A_2(\alpha)_L(2, \mathcal{B})/OPT_L(2, \mathcal{B}) \to 1 + b_2/(3(b_1 + b_2))$ as N goes to infinity.

If $\alpha \geq b_2/3$, consider the following instance L: the first $\lfloor (b_1 - 2b_2/3)N \rfloor$ items with size $1/N$, are followed by item a_1 with size b_2. So $A_2(\alpha)_L(2, \mathcal{B}) \geq b_1 + b_2/3 + b_2 - 1/N$. In an optimal packing, we assign a_1 to B_2, and the remaining items to B_1. $OPT_L(2, \mathcal{B}) = b_1 + b_2$, which follows $A_2(\alpha)_L(2, \mathcal{B})/OPT_L(2, \mathcal{B}) \to 1 + b_2/(3(b_1 + b_2))$ as N tends to infinity. □

By setting $\alpha = b_2/3$, we prove that the competitive ratio of $A_2(\alpha)$ is $1 + b_2/(3(b_1 + b_2))$.

Theorem 3.2. *The competitive ratio of $A_2(\alpha)$ is $1 + b_2/(3(b_1 + b_2))$ if $\alpha = b_2/3$. Moreover the overall competitive ratio is $7/6$.*

Proof. If both bins are light or both are heavy, the packing is optimal. We only need to consider the case that exactly one bin is heavy. From the assumption (2), we have $M_1 + M_2 \geq b_1 + b_2$, where M_i is the load of bin B_i, $i = 1, 2$.

Let $\alpha = b_2/3$. Let T be the excess of the heavy bin and let X be the idle space of the light bin. If $T \leq \alpha$, then $R_{A_2(\alpha)}(2, \mathcal{B}) \leq (T + b_1 + b_2)/(b_1 + b_2) \leq 1 + b_2/(3(b_1 + b_2))$. Now assume that $T > \alpha$. We want to prove $X \leq \alpha$.

Case 1. B_2 is light. $X = b_2 - M_2$. Let a_n be the last item assigned to B_1, which makes the excess over α. $B_2 \neq \emptyset$ and before a_n is assigned B_1 is light and has an idle space at least X. Otherwise, a_n should have been assigned to B_2. It implies that the size of the items in B_2 is larger than $X + \alpha$. Thus $X = b_2 - M_2 < b_2 - (X + \alpha)$. It follows that $X < \alpha$.

Case 2. B_1 is light. $X = b_1 - M_1$. Before the last item is assigned to B_2, B_2 is light and has an idle space at least X. The total size of items in B_2 before the last item is assigned is at most $b_2 - X$. According to the algorithm, $b_2 - X + M_1 > b_1 + \alpha$. It implies that $X \leq \alpha$.

In both cases we have $R_{A_2(\alpha)}(2, \mathcal{B}) \leq (X + M_1 + M_2)/(M_1 + M_2) \leq 1 + b_2/(3(b_1 + b_2))$. It is easy to verify that $R_{A_2(\alpha)}(2, \cdot) = 7/6$. □

Since no on-line algorithm can have an overall competitive ratio less than $7/6$ for two bins [5], $A_2(\alpha)$ is an optimal on-line algorithm for $m = 2$ in terms of the overall competitive ratio, setting $\alpha = b_2/3$.

We will show a lower bound for two bins under the assumption (1). Let $b_1 = kb_2/3 + x$, where $0 \leq x < b_2/3$ and $k \geq 3$ (k is an integer).

Lemma 3.3. *No on-line algorithm can achieve a competitive ratio smaller than*

$$R = \begin{cases} 1 + (b_2/3 - x)/(b_1 + b_2), & \text{if } 0 \leq x \leq b_2/6; \\ 1 + x/(b_1 + b_2), & \text{if } b_2/6 < x < b_2/3. \end{cases}$$

Proof. Let A be any on-line algorithm. Consider the following instance L. The items with size of $b_2/3$ come one by one until $n(b_2) = 2$ or $n(b_1) = k-1$, where $n(b_i)$ is the number of items with size of $b_2/3$ placed into bin B_i by algorithm A for $i = 1, 2$.

Case 1. $n(b_2) = 0, n(b_1) = k-1$. If $0 \le x \le b_2/6$, the next two items have size of $2b_2/3$. Thus $A_L(2, \mathcal{B}) \ge b_1 + b_2 + b_2/3 - x$.

If $b_2/6 < x < b_2/3$, the next item has size of x. If x goes to B_1, two items with size of $2b_2/3$ come. It follows that $A_L(2, \mathcal{B}) \ge b_1 + b_2 + b_2/3$. If x goes to B_2, an item with size of b_2 comes. Then $A_L(2, \mathcal{B}) \ge b_1 + b_2 + x$.

Case 2. $n(b_2) = 1, n(b_1) = k-1$. The next item has size of b_2 and then $A_L(2, \mathcal{B}) \ge b_1 + b_2 + b_2/3$.

Case 3. $n(b_2) = 2, n(b_1) = p$, where $0 \le p \le k-1$.

Subcase 3a) If $p = k-1$, the next item has size of $2b_2/3+x$, and thus $A_L(2, \mathcal{B}) \ge b_1 + b_2 + b_2/3$.

Subcase 3b) If $p = k-2$, we consider two subcases. If $0 \le x \le b_2/6$, the next item has size of x. If it is assigned to B_1, an item with size of b_2 comes. Then $A_L(2, \mathcal{B}) \ge b_1 + b_2 + b_2/3$. If it is assigned to B_2, an item with size of b_2 comes. So $A_L(2, \mathcal{B}) \ge b_1 + b_2 + b_2/3 - x$. If $b_2/6 < x < b_2/3$, the next two items have size of $b_2/3 + x$ and $2b_2/3$. So, $A_L(2, \mathcal{B}) \ge b_1 + b_2 + x$.

Subcase 3c) If $p = k-3$, the next two items have size of $2b_2/3+x/2$. $A_L(2, \mathcal{B}) \ge b_1 + b_2 + b_2/3$.

Subcase 3d) If $p \le k-4$, let $s = \lceil (k-p)/3 \rceil$-1. The next s items have size of b_2. If one of such items is packed into B_2, then $A_L(2, \mathcal{B}) \ge b_1 + b_2 + 2b_2/3$. Otherwise, all the s items go to B_1. The idle space of B_1 becomes $(k-p)b_2/3 + x - (\lceil (k-p)/3 \rceil - 1)b_2$, which is either $b_2/3 + x$ or $2b_2/3 + x$ or $b_2 + x$. Then it is reduced to the above three subcases 3a) -3c).

Note that in any of the above cases, $OPT_L(2, \mathcal{B}) = b_1 + b_2$. Thus, the lemma is proved. □

Lemma 3.3 shows that $R_A(2, \mathcal{B}) \ge 1 + \frac{b_2}{6(b_1+b_2)}$ for any on-line algorithm A. It also shows that algorithm $A_2(\alpha)$ is optimal for the case that $x = 0$ (or $b_1 = kb_2/3$ for some integer k), by setting $\alpha = b_2/3$. Now we consider the three-bin case.

Lemma 3.4. *For any parameter $\alpha > 0$, the competitive ratio of algorithm $A_3(\alpha)$ is at least $1 + b_3/(2(b_1 + b_2 + b_3))$.*

Proof. Let N be a sufficiently large integer. If $\alpha \ge b_3/2$, consider the following instance L. The first $\lfloor b_1 N - 1 \rfloor$ items with size of $1/N$ are followed by an item of size $b_3/2+1/N$. Clearly, $A_3(\alpha)_L(3, \mathcal{B}) \ge b_1+b_2+3b_3/2-1/N$ and $OPT_L(3, \mathcal{B}) \le b_1 + b_2 + b_3 + 1/N$. It follows that $A_3(\alpha)_L(3, \mathcal{B})/OPT_L(3, \mathcal{B}) \to 1 + b_3/(2(b_1 + b_2 + b_3))$ as N tends to infinity.

If $\alpha < b_3/2$, consider the following instance L. $\lfloor b_1 N - 1 \rfloor$ items with size of $1/N$ are followed by item a_1 with size of $\alpha+1/N$. Then $\lfloor (b_2 - b_3/2)N \rfloor + 1$ items with size of $1/N$ are coming, which are followed by the last two items a_2, a_3 with

size of $b_3/2$, $b_3-\alpha-1/N$, respectively. Algorithm $A_3(\alpha)$ assigns the first $\lfloor b_1N-1\rfloor$ items together with a_1 to B_1, $\lfloor(b_2-b_3/2)N+1\rfloor$ items with size $1/N$ to B_2, and a_2 and a_3 to B_3. It shows $A_3(\alpha)_L(3,\mathcal{B}) \geq b_1+b_2+3b_3/2-2/N$. In an optimal packing, we can assign a_1 and a_3 to B_3, the first $\lfloor b_1N-1\rfloor$ items with size $1/N$ to B_1 and the remaining items to B_2. It follows that $OPT_L(3,\mathcal{B}) \leq b_1+b_2+b_3+1/N$. Then $A_3(\alpha)_L(3,\mathcal{B})/OPT_L(3,\mathcal{B}) \to 1+b_3/(2(b_1+b_2+b_3))$, as N tends to infinity. □

In the following, we prove that the competitive ratio of algorithm $A_3(\alpha)$ can reach the lower bound $1+b_3/(2(b_1+b_2+b_3))$ by setting $\alpha = b_3/2$.

Theorem 3.5. *The competitive ratio of the algorithm $A_3(\alpha)$ is $1+b_3/(2(b_1+b_2+b_3))$ if $\alpha = b_3/2$.*

Proof. Let $\alpha = b_3/2$. In the packing given by $A_3(\alpha)$, let M_i be the total size of the items assigned to bin B_i. If all the bins are heavy or all the bins are light, the algorithm gives an optimal packing. Hence, we only need to consider the following cases.

Case 1. Only one bin is heavy. If the excess of the heavy bin is at most α, we can get the competitive ratio $1+b_3/(2(b_1+b_2+b_3))$ immediately. Before the last item of the heavy bin is assigned, all the three bins are light and the heavy bin has the largest idle space at the moment.

Subcase 1a) B_1 is heavy. Then $M_1 > b_1+\alpha$. Note that $M_2+M_3 > b_2+\alpha$. We have

$$\begin{aligned} R_{A_3(\alpha)}(3,\mathcal{B}) &\leq (M_1+b_2+b_3)/(M_1+M_2+M_3) \\ &= 1+(b_2+b_3-M_2-M_3)/(M_1+M_2+M_3) \\ &\leq 1+b_3/(2(b_1+b_2+b_3)) \end{aligned}$$

Subcase 1b) B_2 is heavy. $M_2 > b_2+b_3/2$ and $M_1+M_3 > b_1+b_3/2$.

$$\begin{aligned} R_{A_3(\alpha)}(3,\mathcal{B}) &\leq (M_2+b_1+b_3)/(M_1+M_2+M_3) \\ &= 1+(b_1+b_3-M_1-M_3)/(M_1+M_2+M_3) \\ &\leq 1+b_3/(2(M_1+M_2+M_3)) \\ &\leq 1+b_3/(2(b_1+b_2+b_3)) \end{aligned}$$

Subcase 1c) B_3 is heavy. $M_3 > b_3+b_3/2$. Let X be the total idle space of M_1 and M_2. Then $X = b_1+b_2-(M_1+M_2)$. Let Y_3 be the load in B_3 before the last item is assigned. Thus

$$b_3-Y_3 \geq b_2-M_2,\ b_3-Y_3 \geq b_1-M_1.$$

We have $X = b_1+b_2-(M_1+M_2) \leq 2(b_3-Y_3)$. On the other hand,

$$Y_3+M_1 \geq b_1+b_3/2,\ Y_3+M_2 \geq b_2+b_3/2.$$

We have $X = b_1+b_2-(M_1+M_2) \le 2Y_3-b_3$. So, $X \le \min\{2(b_3-Y_3), 2Y_3-b_3\} \le b_3/2$. Therefore,

$$\begin{aligned} R_{A_3(\alpha)}(3,\mathcal{B}) &\le (X + M_1 + M_2 + M_3)/(M_1 + M_2 + M_3) \\ &\le 1 + b_3/(2(b_1 + b_2 + b_3)). \end{aligned}$$

Case 2. Only one bin is light. If the idle space X of the light bin is at most α or the total excess of the two heavy bins is at most α, we have the competitive ratio immediately. We only need to consider the case that $X > \alpha$ and the total excess exceeds α. However, We will show that it is impossible. Before considering the following two cases, we assume that $M_i > 0$ for $i = 1, 2, 3$. Otherwise, the total excess is at most α.

Subcase 2a) B_3 is light. $M_3 = b_3 - X < \alpha = b_3/2$. Let Y_i be the load of B_i immediately before it becomes heavy, for $i = 1, 2$. Then $b_i - Y_i \ge X$; otherwise the last item, which makes the total excess of B_1 and B_2 over α, would have been assigned to bin B_3. Thus all the items in B_3 should have been assigned to B_i by the algorithm. It is a contradiction.

Subcase 2b) B_3 is heavy. Recall that any item size is at most b_3. There are at least two items in B_3. Assume that a_k is the last item assigned to B_3. Those items assigned to B_3 have size at least $X > b_3/2$. Thus the idle space of B_3 before a_k is assigned is less than X, which means that a_k would have not been assigned to B_3. It is a contradiction. □

From Theorems 3.2 and 3.5 we realize that the competitive ratio of algorithm $A_m(\alpha)$ for unequal bins is better than that for equal bins.

4 Without the Assumption on Item Sizes

In this section, we will discuss the problem without the assumption (1) for the two-bin case.

Lemma 4.1. *Without the assumption (1), no deterministic on-line algorithms can achieve an overall competitive ratio lower than* $6/5$ *for two bins.*

Proof. Let $b_1 = 3/2$ and $b_2 = 1$. The first item a_1 has size of $1/2$. If a_1 is assigned to bin B_2, two items a_2, a_3 with size of 1 are coming. If a_1 is assigned to bin B_1, item a_2 with size of $3/2$ is coming. For both cases, the overall competitive ratio is at least $6/5$. □

Lemma 4.2. *For any on-line algorithm A the competitive ratio*

$$R_A(2,\mathcal{B}) \ge \begin{cases} 1 + (4b_2/3 - b_1)/(b_1 + b_2), & \text{if } b_1 \le 7b_2/6, \\ 1 + (b_1 - b_2)/(b_1 + b_2), & \text{if } 7b_2/6 < b_1 < 4b_2/3, \\ 1 + b_2/(3(b_1 + b_2)), & \text{if } b_1 \ge 4b_2/3. \end{cases}$$

Proof. Let A be any on-line algorithm. Consider first the case that $b_1 \ge 4b_2/3$ with the following instance L. The first two items have size of $b_2/3$. If both

items go to B_1, the next item has size of b_1; if both items go to B_2, two items with size of $2b_2/3$ and $b_1 - b_2/3$, respectively, are coming; if the first two items go to different bins, the next item has size of b_1. For any of the above cases, $A_L(2, \mathcal{B}) \geq b_1 + 4b_2/3$ and the optimal value $OPT_L(2, \mathcal{B}) = b_1 + b_2$. It implies that the lemma is true.

Now turn to the case that $b_1 < 4b_2/3$. Let $b_1 = b_2 + x$, where $0 \leq x < b_2/3$. The lemma follows from Lemma 3.3 by setting $k = 3$. □

We consider on-line algorithms. Assume that all items have size of at most b_1 ($b_1 \geq b_2$).

Theorem 4.3. *The competitive ratio of the list scheduling algorithm LS for two bins is* $1 + \min\{b_2, b_1/2\}/(b_1 + b_2)$.

Proof. We first show the upper bound. Without loss of generality, we consider the case only one bin is heavy.

Case 1. $b_1/2 \geq b_2$. Since the LS algorithm assigns items to the bin with largest idle space. Recall that the total size of items is not less than the total size of bins, we get that the idle space is at most b_2. Then, $LS_L(2, \mathcal{B}) \leq \sum p_i + b_2$, $OPT_L(2, \mathcal{B}) \geq \sum p_i \geq b_1 + b_2$, which implies $R_{LS}(2, \mathcal{B}) \leq 1 + b_2/(b_1 + b_2)$.

Case 2. $b_1/2 < b_2$. Let X be the idle space and T be the excess. We only consider the case that both X and T are greater than $b_1/2$. Otherwise, it follows that $R_{LS}(2, \mathcal{B}) \leq 1 + b_1/(2(b_1 + b_2))$. Let p_n be the size of the last item assigned to the bin which has an excess over $b_1/2$. Recall that all items have size of at most b_1. Let Y_i be the load of bin B_i before the last item is assigned. Then $b_i - Y_i \geq X > b_1/2$, since $p_n \leq b_1$. It implies that either X or T is at most $b_1/2$.

The following simple instance shows that the bound is tight. Consider three items with sizes of $\min\{b_2, b_1/2\}$, $\max\{0, b_2 - b_1/2\}$, and b_1, respectively. The optimal value is $b_1 + b_2$ while LS costs $b_1 + b_2 + \min\{b_2, b_1/2\}$. □

In the following, we present an on-line algorithm to improve the upper bound in some cases.

Algorithm *A2*:

- If $b_1 \leq 4b_2/3$, apply algorithm $A_2(\alpha)$ by setting $\alpha = b_2/3$;
- if $4b_2/3 < b_1 \leq 2b_2$, apply $A_2(\alpha)$ by setting $\alpha = b_1 - b_2$;
- if $b_1 > 2b_2$, apply algorithm LS.

Theorem 4.4. *The competitive ratio of algorithm A2 is*

$$R_{A2}(2, \mathcal{B}) \leq \begin{cases} 1 + b_2/(3(b_1 + b_2)), & \text{if } b_1 \leq 4b_2/3, \\ 1 + (b_1 - b_2)/(b_1 + b_2), & \text{if } 4b_2/3 < b_1 \leq 2b_2, \\ 1 + b_2/(b_1 + b_2), & \text{if } b_1 > 2b_2. \end{cases}$$

Proof. Consider the following cases:

Case 1. $b_1 \leq 4b_2/3$. We use $A_2(\alpha)$, by setting $\alpha = b_2/3$. The proof is similar as the one of Theorem 3.2.

Case 2. $4b_2/3 < b_1 \leq 2b_2$. We only need to consider the case that exactly one bin is heavy, the excess of the heavy bin is over $b_1 - b_2$, and the idle space of the light bin is larger than $b_1 - b_2$. However, we will show this case is impossible. Before the last item a_n, which makes the excess over $b_1 - b_2$, is assigned, if B_2 is empty, then the excess is at most $b_1 - b_2$ after assigning a_n. Assume that B_2 is not empty before a_n is assigned. Note that B_1 has an idle space larger than $b_1 - b_2$ before a_n is assigned. It means that the total size of items in B_2 is more than $2(b_1 - b_2)$. Then the idle space of bin B_2 is at most $b_2 - 2(b_1 - b_2) < (b_1 - b_2)$. In this case a_n is assigned to B_1. B_2 is a light bin and the idle space is less than $b_1 - b_2$. It is a contradiction.

Case 3. $b_1 > 2b_2$. It follows from Theorem 4.3. □

Final Remarks. There are many open questions. Although we have proved that the algorithm $A_2(\alpha)$ is optimal in terms of the overall competitive ratio, it is still open to find a best possible on-line algorithm for two bins in terms of the competitive ratio. It is interesting to design an algorithm with a better competitive ratio than LS algorithm for any number of bins. The problem without the assumption (1) becomes more difficult. We only proved the competitive ratio of LS for the two-bin case. Any improvement on the lower bounds is also of interest.

References

1. N. Alon, Y. Azar, G.J. Woeginger, and T. Yadid, Approximation schemes for scheduling on parallel machines, *Journal of Scheduling* **1** (1998), 55-66.
2. Y. Azar and O. Regev, On-line bin-stretching, *Theoretical Computer Science* **268** (2001), 17-41.
3. E.G. Coffman and G.S. Lueker, Approximation algorithms for extensible bin packing, *Proceedings of the twelfth Annual ACM-SIAM Symposium on Discrete Algorithms* (2001), 586-588.
4. P. Dell'Olmo, H. Kellerer, and M.G. Speranza, Z. Tuza, A 13/12 approximation algorithm for bin packing with extendable bins, *Information Processing Letters* **65** (1998), 229-233.
5. P. Dell'Olmo and M.G. Speranza, Approximation algorithms for partitioning small items in unequal bins to minimize the total size, *Discrete Applied Mathematics* **94** (1999), 181-191.
6. L. Epstein, Bin stretching revisited, *Acta Informatica* **39** (2003), 97-117.
7. M.R. Garey and D.S. Johnson, Computers and Intractability: A guide to the Theory of NP-completeness, W.H.Freeman, San Francisco, 1979.
8. R. Graham, Bounds for certain multiprocessing anomalies, *Bell System Technical Journal* **45** (1966), 1563-1581.
9. J. Sgall, On-line scheduling - A survey, Online Algorithms: The State of the Art, eds. A. Fiat and G. J. Woeginger, *Lecture Notes in Comput. Sci.* **1442**, 196-231, Springer-Verlag, 1998.

10. M.G. Speranza and Z. Tuza, On-line approximation algorithms for scheduling tasks on identical machines with extendable working time, *Annals of Operations Research* **86** (1999), 494-506.
11. G.J. Woeginger, When does a dynamic programming formulation guarantee the existence of an FPTAS? In *Proceedings of the tenth Annual ACM-SIAM Symposium on Discrete Algorithms* (1999), 820-829.
12. D. Ye and G. Zhang, On-line scheduling with extendable working time on a small number of machines, *Information Processing Letters* **85** (2003), 171-177.

Energy Consumption in Radio Networks: Selfish Agents and Rewarding Mechanisms

Christoph Ambühl[1,⋆], Andrea E.F. Clementi[1], Paolo Penna[2,⋆⋆], Gianluca Rossi[1,⋆⋆], and Riccardo Silvestri[3]

[1] Dipartimento di Matematica, Università di Roma "Tor Vergata"
{ambuehl,clementi,rossig}@mat.uniroma2.it
[2] Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Università di Salerno
penna@dia.unisa.it
[3] Dipartimento di Informatica, Università di Roma "La Sapienza"
silver@dsi.uniroma1.it

Abstract. We consider the *range assignment* problem in ad-hoc wireless networks in the context of *selfish agents*: a network manager aims in assigning transmission ranges to the stations so to achieve a suitable network with a minimal *overall energy*; stations are not directly controlled by the manager and may refuse to transmit with a certain transmission range because this results in a power consumption proportional to that range.
We investigate the existence of payment schemes which induce the stations to cooperate with a network manager computing a range assignment, that is, *truthful mechanisms* for the range assignment problem.

1 Introduction

One of the main benefits of ad-hoc wireless networks relies in the possibility of communicating without any fixed infrastructure. Indeed, it just consists of a set of stations which communicate with each other via radio transmitters and receivers. Due to the limited power of the stations, *multi-hop* transmissions are in general unavoidable. That is, instead of sending the message directly from the source station to the sink station, the message is sent via intermediate stations. Another benefit of multi-hop transmissions is that they reduce the overall energy required by the communication.

Assigning transmission powers to stations which (i) guarantee a "good" communication between stations, and (ii) minimize the overall power consumption of the network gives rise to interesting algorithmic questions. In particular, these two aspects yield a class of fundamental optimization problems, denoted as *range assignment* problems, that has been the subject of several works in the area of wireless network theory [7, 5, 4].

⋆ Partially supported by the European Union under the RTN Poject ARACNE.
⋆⋆ Partially supported by the European Union under the IST FET Poject CRESCCO.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 248–251, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Unfortunately, the inherent *self-organized* nature of ad-hoc networks can complicate these problems even more. Since there is no central authority controlling the stations, we have to assume that they will act as selfish agents. That is, they will always try to optimize their own benefit, without considering the social cost of their behavior. This is exploited in the following model.

2 The Model

Range assignments without selfish stations. The instance of the range assignment problem consists of a complete weighted digraph $G(S, S \times S)$ and a graph property π. The set S denotes the set of stations. For $i, j \in S$, let C^i_j be the weight of the arc from station i to station j. This can be interpreted as the cost for sending a message from station i to station j.

The goal is to find the cheapest subgraph $G(S, E)$ that satisfies the graph property π. In general, we just write E to denote the subgraph.

The cost of E, also called communication graph, differs slightly from the one for wired networks. Let $E^i \subseteq E$ be the set of links emanating from station i and let $C^i = \{C^i_1, \ldots, C^i_n\}$ be the cost vector or them. Since station i can send a message to all its out-neighbors, provided it uses enough power to reach all of them, its cost depend only on the most expensive arc it has to provide. The cost of station i for maintaining the communication graph E, denoted by $\texttt{cost}^i$, is thus

$$\texttt{cost}^i(C^i, E) := \max_{j\,:\,(i,j)\in E} C^i_j.$$

Hence, the total cost of the range assignment E is

$$\texttt{cost}(\{C^1, \ldots, C^n\}, E) = \sum_{i \in S} \texttt{cost}^i(C^i, E).$$

Range assignment with selfish stations. We consider each station as a selfishly acting agent that *privately knows* part of the input: station i privately knows C^i that the manager must use for the computation of a feasible solution.

We assume that the range assignment is chosen by a network manager. In order to choose a low-cost solution, the manager needs information about the cost of the connections. So, in the first phase, every station i sends a vector $D^i = \langle D^i_1, \ldots, D^i_n \rangle$ to the network manager, where D^i_j denotes its *declared cost* for maintaining its link to station j. In an ideal world, we could assume that station i just sends the true values C^i. But since we assume that the station act selfishly, we have to assume that they lie to the network manager by declaring $D^i \neq C^i$. In the second phase, the range assignment is computed on the declared values and then implemented by informing the stations about E.

However, in order to be successful in this model, one needs to convince the stations to always tell the truth. This can be achieved by a so-called *mechanism*. A mechanism for a range assignment problem is a pair (ALG, P), where ALG is an algorithm that, on input $\mathcal{D} = \{D^1, \ldots, D^n\}$, returns a feasible range assignment $E = \text{ALG}(\mathcal{D})$ and a payment vector $P = P(\mathcal{D}, E) = \{P^1, \ldots, P^n\}$, consisting of

a payment for each station. Hence, agents, being selfish, will try to maximize their utility

$$U^i = P^i(\mathcal{D}, E) - \mathtt{cost}^i(C^i, E),\ i = 1, \ldots, n.$$

A mechanism (ALG, P) is called *truthful* if any agent i can always maximize her utility by declaring the truth, i.e., when $D^i = C^i$. This has to hold for all possible declarations of the other agents.

3 Truthful VCG Mechanisms

The theory of mechanism design dates back to the seminal papers by Vickrey [10], Clarke [3] and Groves [6]. Their celebrated *VCG mechanism* is still the prominent technique to derive truthful mechanisms for many problems (e.g., shortest path, minimum spanning tree, etc.). In particular, when applied to combinatorial optimization problems (see e.g., [8, 9]), the VCG mechanisms guarantee the truthfulness under the hypothesis that the optimization function is *utilitarian*, that is, the optimization function is equal to the sum of the single agents' valuations, and that ALG computes always the optimum. The second condition can be weakened to the following [9].

Property I: An algorithm ALG has property I if the solution it returns is optimal among all the solutions in its output set.

Let us denote by $\mathcal{D}^{-i}$ the set of declarations $\mathcal{D} \setminus D^i$. The payment scheme is of the form

$$P^i = -\sum_{j \neq i} \mathtt{cost}^j(D^j, E) + h^i(\mathcal{D}^{-i}),$$

where $h^i(\cdot)$ is any function independent from D^i. Intuitively, these mechanisms achieve truthfulness by relating the utility of an agent with the total cost of the solution chosen by the mechanism: The better the solution the higher the utility. In order to formalize this statement let us consider the utility of agent i.

$$\begin{aligned} U^i &= P^i - \mathtt{cost}^i(C^i, E) \\ &= h^i(\mathcal{D}^{-i}) - \mathtt{cost}^i(C^i, E) - \sum_{j \neq i} \mathtt{cost}^j(D^j, E) \\ &= h^i(\mathcal{D}^{-i}) - \mathtt{cost}(\langle \mathcal{D}^{-i}; C^i \rangle, E). \end{aligned}$$

Since $h^i(\mathcal{D}^{-i})$ is independent of D^i, agent i tries to minimize $\mathtt{cost}(\langle \mathcal{D}^{-i}; C^i \rangle, E)$. Hence, his declaration should be chosen such that the algorithm returns a solution $\tilde{E}$ which minimizes $\mathtt{cost}(\langle \mathcal{D}^{-i}; C^i \rangle, E)$. This can be achieved simply by declaring the truth, assuming that the mechanism has property I.

Many range assignment problems are known to be NP-hard or even $\log APX$-hard. We therefore cannot expect to find an algorithm that always returns the optimal solution. Therefore, we have to go for property I. The main problem we are faced with in the area of truthful mechanisms for range assignment problems thus is to find good approximation algorithms ALG with property I.

4 Previous Related Work and Open Problems

Range assignment problems are widely studied. For a survey, the reader is referred to [4]. A very interesting version is the Euclidean version, where the stations are point in the Euclidean plane and the cost to send a message from i to j is $C_j^i = \text{dist}(i,j)^\alpha$, where $\text{dist}(i,j)$ is the Euclidean distance between i and j and $\alpha \geq 1$ is a constant. This model admits a polynomial-time 2-approximation algorithm [7].

The only case in which a mechanism is known is the case $\alpha = 1$. There, the $\frac{\sqrt{5}+1}{2}$-approximation algorithm by Călinescu and Zaragoza [2] can be turned into a mechanism [1]. Also in [1], it was observed that the approximation ratio of this algorithm was improved to 1.5.

Several interesting problems are open in this area. First of all, in would be nice to design mechanisms for other types of range assignment problems. Furthermore, one could think of different models of privacy. For instance, one can assume that the protocol has a partial knowledge of the network topology. Finally, even though the VCG method is the major technique in order to obtain efficient truthful rewarding mechanism, a fundamental future research is to develop alternative methods to manage selfish behavior in the context of energy consumption in wireless networks.

References

1. C. Ambühl, A. E. F. Clementi, P. Penna, G. Rossi, and R. Silvestri. Energy consumption in radio networks: Selfish agents and rewarding mechanisms. In *Proc. of 10th SIROCCO*, pages 1–16, 2003.
2. G. Calinescu and F.Zaragoza. Unpublished manuscript, 2002.
3. E.H. Clarke. Multipart Pricing of Public Goods. *Public Choice*, pages 17–33, 1971.
4. A.E.F. Clementi, G. Huiban, P. Penna, G. Rossi, and Y.C. Verhoeven. Some Recent Theoretical Advances and Open Questions on Energy Consumption in Ad-Hoc Wireless Networks. In *Proceedings of the 3rd Workshop on Approximation and Randomization Algorithms in Communication Networks (ARACNE)*, pages 23–38, 2002. Also available in http://www.mat.uniroma2.it/∼rossig/.
5. A. Ephremides, G.D. Nguyen, and J.E. Wieselthier. On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 585–594, 2000.
6. T. Groves. Incentive in Teams. *Econometrica*, 41:617–631, 1973.
7. L. M. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc. Power Consumption in Packet Radio Networks. *Theoretical Computer Science*, 243:289–305, 2000.
8. N. Nisan and A. Ronen. Algorithmic Mechanism Design. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 129–140, 1999.
9. A. Ronen. *Solving Optimization Problems Among Selfish Agents.* PhD thesis, Hebrew University in Jerusalem, 2000.
10. W. Vickrey. Counterspeculation, Auctions and Competitive Sealed Tenders. *Journal of Finance*, pages 8–37, 1961.

Power Consumption Problems in Ad-Hoc Wireless Networks*

Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos

Computer Technology Institute and
Dept. of Computer Engineering and Informatics
University of Patras, 26500 Rio, Greece

Wireless networks have received significant attention during the recent years. Especially, *ad hoc wireless networks* emerged due to their potential applications in battlefield, emergency disaster relief, etc. [13]. Unlike traditional wired networks or cellular wireless networks, no wired backbone infrastructure is installed for ad hoc wireless networks.

A node (or station) in these networks is equipped with an omnidirectional antenna which is responsible for sending and receiving signals. Communication is established by assigning to each station a transmitting power. In the most common power attenuation model [13], the signal power falls as $1/r^\alpha$, where r is the distance from the transmitter and α is a constant which depends on the wireless environment (typical values of α are between 1 and 6). So, a transmitter can send a signal to a receiver if $\frac{P}{d(s,t)} \geq \gamma$ where P_s is the power of the transmitting signal, $d(s,t)$ is the Euclidean distance between the transmitter and the receiver, and γ is the receiver's power threshold for signal detection which is usually normalized to 1. So, communication from a node s to another node t may be established either directly if the two nodes are close enough and s uses adequate transmitting power, or by using intermediate nodes. Observe that due to the nonlinear power attenuation, relaying the signal between intermediate nodes may result in smaller power consumption.

A crucial issue in ad hoc wireless networks is to support communication patterns that are typical in traditional networks. These include broadcasting, multicasting, and gossiping (all–to–all communication). Since establishing a communication pattern strongly depends on the power levels, the important engineering question to be solved is to guarantee a desired communication pattern minimizing the total power consumption. In this work, we consider a series of power consumption problems which we formulate below.

Consider a complete directed graph $G = (V, E)$, where $|V| = n$, with a non-negative edge cost function $c : E \rightarrow R^+$. Given a non-negative node weight assignment $w : V \rightarrow R^+$, the *transmission graph* G_w is the directed graph defined as follows. It has the same set of nodes as G and a directed edge (u, v) belongs to G_w if the weight assigned to node u is at least the cost of the edge (u, v), i.e., $w(u) \geq c(u, v)$. Intuitively, the weight assignment corresponds to the

* This work was partially supported by the European Union under IST FET Project ALCOM–FT, IST FET Project CRESCCO, and RTN Project ARACNE.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 252–255, 2004.
© Springer-Verlag Berlin Heidelberg 2004

power levels at which each node operates (i.e., transmits messages) while the cost between two nodes indicates the minimum power required to send messages from one node to the other. Usually, the edge cost function is symmetric (i.e., $c(u,v) = c(v,u)$). In ideal cases, we can assume that the nodes of the graph are points in a Euclidean space and the cost of an edge between two nodes is the Euclidean distance of the corresponding points raised to a constant power. Asymmetric edge cost functions can be used to model medium abnormalities or batteries with different power levels [12].

The problems we study can be generally stated as follows. Given a complete directed graph $G = (V, E)$, where $|V| = n$, with non-negative edge costs $c : E \to R^+$, find a non-negative node weight assignment $w : V \to R^+$ such that the transmission graph G_w maintains a connectivity property and the sum of weights is minimized. Such a property is defined by a requirement matrix $R = (r_{ij}) \in \{0, 1\}$ where r_{ij} is the number of directed paths required in the transmission graph from node v_i to node v_j.

In Minimum Energy Multicast Tree (MEMT), the connectivity property is defined by a root node v_0 and a set of nodes $D \subseteq V - \{v_0\}$ such that $r_{ij} = 1$ if $i = 0$ and $v_j \in D$ and $r_{ij} = 0$, otherwise. The Minimum Energy Broadcast Tree (MEBT) is the special case of MEMT with $D = V - \{v_0\}$. Liang [12] presents an intuitive reduction of any instance of MEMT to an instance of Directed Steiner Tree (DST). An $O(|D|^\epsilon)$-approximation algorithm for any $\epsilon > 0$ then follows by using an approximation algorithm for DST[1]. The authors show in [4] that MEMT is also at least as hard as DST (this is also observed in [2]). Using a recent inapproximability result for DST, we obtain an $O(\ln^{2-\epsilon} n)$ inapproximability result for MEMT. Hence, improvements on the approximability bounds of MEMT strongly depend on new results for DST. It is strongly believed that polylogarithmic approximation algorithms for the latter do exist. In symmetric graphs, there exists an $O(\ln |D|)$-approximation algorithm. This is shown in [4] where instances of MEMT in symmetric graphs are reduced to instances of Node-Weighted Steiner Tree, a problem which is known to be approximable within a logarithmic factor. Note that this result is asymptotically optimal since it can be easily seen that MEMT in symmetric graphs is at least as hard to approximate as Set Cover [5]. MEBT is easier and can be approximated within $O(\ln n)$ as it was recently shown in [4] and [2]. In [4], the authors show that the corresponding instance of DST has some very good properties and can be approximated within a logarithmic factor. Calinescu et al. [2] achieve a similar approximation bound by a simpler algorithm which constructs a tree incrementally. Sophisticated set-covering techniques are used for the analysis. Constant approximation algorithms for the special case of MEBT where the nodes of the input graph correspond to points in the Euclidean plane are presented in [5, 14]. The special case of MEBT where the nodes are points in a line can be solved in polynomial time [3, 6].

[1] Due to lack of space, references to known results for combinatorial problems which are crucial in the proofs of the results presented have been omitted from this survey. The interested reader may see the papers cited here and the references therein.

In Minimum Energy Steiner Subgraph (MESS), the requirement matrix is symmetric. Alternatively, we may define the problem by a set of nodes $D \subseteq V$ partitioned into p disjoint subsets $D_1, D_2, ..., D_p$. The entries of the requirement matrix are now defined as $r_{ij} = 1$ if $v_i, v_j \in D_k$ for some k and $r_{ij} = 0$, otherwise. The Minimum Energy Subset Strongly Connected Subgraph (MESSCS) is the special case of MESS with $p = 1$ while the Minimum Energy Strongly Connected Subgraph (MESCS) is the special case of MESSCS with $D = V$ (i.e., the transmission graph is required to span all nodes of V and to be strongly connected). Although the approximability of MESS is yet unknown, MESSCS has the same approximability properties as MEMT, i.e., it can be approximated within $O(|D|^{\epsilon})$ but cannot be approximated within $O(\ln^{2-\epsilon} n)$. Whether there are polylogarithmic approximations for this problem is an interesting open problem and strongly depends on new results for DST. MESCS can be approximated within a logarithmic factor using an algorithm for MEBT as a subroutine. This result is asymptotically optimal. In symmetric graphs, MESS, MESSCS and MESCS have constant approximation algorithms using algorithms for network design problems as a subroutine [4] (see also [11]). The respective bounds are a 2-approximation algorithm for MESCS using optimal Spanning Tree solutions (this was first proved by Kirousis et al. in [10]), a 2ρ-approximation algorithm for MESSCS using ρ-approximate Steiner Tree solutions, and a 4-approximation algorithm for MESS using a 2-approximation algorithm for Steiner Forest. Approximation schemes are unlikely to exist for these problems in symmetric graphs since inapproximability bounds have been proved for MESCS [8] using approximation-preserving reductions from Vertex Cover in bounded-degree graphs. The special case of MESCS where the nodes are points in a line can be solved in polynomial time [7].

The authors of [1] study MESCS under the extra requirement that the transmission graph contains a bidirected subgraph which maintains the connectivity requirements of MESCS. By adding this extra requirement to MESS and MESSCS, we obtain the bidirected MESS and bidirected MESSCS, respectively. In [4], it is shown that instances of bidirected MESS can be reduced to instances of Node-Weighted Steiner Forest, a problem for which logarithmic approximations exist. As corollaries, we obtain $O(\ln |D|)$- and $O(\ln n)$-approximation algorithms for bidirected MESSCS and bidirected MESCS, respectively. A slightly inferior logarithmic approximation bound for bidirected MESCS is presented in [2] using different techniques. These results asymptotically match the inapproximability result for bidirected MESCS of Althaus et al. [1]. For symmetric graphs, the positive and negative results for MESS, MESSCS, and MESCS hold for their bidirected versions as well. Better approximation algorithms for bidirected MESCS are presented in [1].

We have surveyed results on power consumption problems where the connectivity requirements are defined by 0-1 requirement matrices. A natural extension is to consider matrices with non-negative integer entries r_{ij} denoting that at least r_{ij} node-disjoint paths are required from node v_i to node v_j. This extension leads

to interesting power consumption problems for fault torelant wireless communication. Some recent results on such problems can be found in [9, 11].

References

1. E. Althaus, G. Călinescu, I. Măndoiu, S. Prasad, N. Tchervenski, and A. Zelikovsky. Power Efficient Range Assignment in Ad-Hoc Wireless Networks. In *Proc. of the IEEE Wireless Communications and Networking Conference (WCNC '03)*, IEEE Computer Society Press, pp. 1889-1894, 2003.
2. G. Călinescu, S. Kapoor, A. Olshevsky and A. Zelikovsky. Network Lifetime and Power Assignment in Ad-Hoc Wireless Networks. In *Proc. of the 11th Annual European Symposium on Algorithms (ESA '03)*, LNCS 2832, Springer, 2003.
3. I. Caragiannis, C. Kaklamanis and P. Kanellopoulos. New Results for Energy-Efficient Broadcasting in Wireless Networks. *In Proc. of the 13th Annual International Symposium on Algorithms and Computation (ISAAC '02)*, LNCS 2518, Springer, pp. 332-343, 2002.
4. I. Caragiannis, C. Kaklamanis and P. Kanellopoulos. Energy-Efficient Wireless Network Design. *In Proc. of the 14th Annual International Symposium on Algorithms and Computation (ISAAC '03)*, 2003, to appear.
5. A. E. F. Clementi, P. Crescenzi, P. Penna, G. Rossi, and P. Vocca. On the Complexity of Computing Minimum Energy Consumption Broadcast Subgraphs. In *Proc. of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS '01)*, LNCS 2010, Springer, pp. 121–131, 2001.
6. A. E. F. Clementi, M. Di Ianni, R. Silvestri. The Minimum Broadcast Range Assignment Problem on Linear Multi-Hop Wireless Networks. *Theoretical Computer Science*, 1-3(299): 751-761, 2003.
7. A. E. F. Clementi, P. Penna, A. Ferreira, S. Perennes, and R. Silvestri. The Minimum Range Assignment Problem on Linear Radio Networks. *Algorithmica*, 35(2): 95-110, 2003.
8. A. E. F. Clementi, P. Penna, and R. Silvestri. Hardness Results for the Power Range Assignment Problem in Packet Radio Networks. In *Proc. of Randomization, Approximation, and Combinatorial Optimization (RANDOM/APPROX '99)*, LNCS 1671, Springer, pp. 197–208, 1999.
9. M. Hajiaghayi, N. Immorlica and V. Mirrokni. Power Optimization in Fault-Tolerant Topology Control Algorithms for Wireless Multi-hop Networks. In *Proc. of the 9th Annual International Conference on Mobile Computing and Networking (MOBICOM '03)*, pp. 300-312, 2003.
10. L. M. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc. Power Consumption in Packet Radio Networks. *Theoretical Computer Science*, 243(1-2):289-305, 2000.
11. E. Lloyd, R. Liu, M. Marathe, R. Ramanathan and S.S. Ravi. Algorithmic Aspects of Topology Control Problems for Ad hoc Networks. In *Proc. of 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC '02)*, pp. 123-134, 2002.
12. W. Liang. Constructing Minimum-Energy Broadcast Trees in Wireless Ad Hoc Networks. In *Proc. of 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC '02)*, pp. 112-122, 2002.
13. T. S. Rappaport. Wireless Communications: Principles and Practices. *Prentice Hall*, 1996.
14. P.-J. Wan, G. Călinescu, X.-Y. Li, and O. Frieder. Minimum-Energy Broadcasting in Static Ad Hoc Wireless Networks. *Wireless Networks*, 8(6):607-617, 2002.

A Combinatorial Approximation Algorithm for the Multicommodity Flow Problem*

David Coudert, Hervé Rivano, and Xavier Roche

cnrs-i3s-inria Mascotte – France Telecom R&D – Éns Lyon

Abstract. This work is motivated by the need for approximation algorithms for the integral multicommodity flow problem which arise in numerous optimization scenarios, including the design of telecommunication networks. We improve on one of the most efficient known combinatorial approximation algorithm for fractional multicommodity flow by using an incremental approach. This approach is validated by experimental results, which show a significant speed-up.

Keywords: Multicommodity flow, combinatorial approximation, dynamic shortest paths, incremental algorithm.

The multicommodity flow problem is useful in numerous applications, especially when the issue is to compute paths for entities that are concurrent for some resources. In these cases, the resources are modeled by capacities on the edges of the graph bearing all possible routings.

If the multicommodity flow is constrained to be integer, the problem is $\mathcal{NP}$-hard. Nevertheless, Raghavan [Rag94] proposed a randomized algorithm constructing a multicommodity flow based on the randomized rounding of the linear relaxation of the problem, namely the fractional multicommodity flow. Recently an improvement of this algorithm has been proposed [CR02]. This algorithm computes a better approximation of the integer multicommodity flow, but at the cost of solving a fractional multicommodity flow for each commodity in an iterative process. Solving the fractional multicommodity flow efficiently is therefore a crucial issue.

Given that the fractional multicommodity flows is used to compute an approximated integral solution, there is no real need for the use of optimal fractional flows. Indeed, the approximation yielded by the randomized rounding process is an additive gap of order the square root of the fractional capacities [Rag94,CR02]. If these fractional capacities are $(1+\epsilon)$-approximations of the optimal capacities, the magnitude of the final gap will not change dramatically. It is therefore natural to exploit this freedom in order to consequently speed up the integer multicommodity flow approximation algorithm. A first article [GK98] proposes a combinatorial algorithm computing a fractional multicommodity flow $(1+\epsilon)$-approximation in time $O(\epsilon^{-2}km^2n^2)$, where k is the number of commodities, n and m are the number of vertices and edges of the flow network. It is

* Work funded by European project fet crescco and action color dynamic.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 256–259, 2004.
© Springer-Verlag Berlin Heidelberg 2004

then possible to avoid the linear program solvers, much more costly to run. This algorithm has been improved by Fleischer [Fle00], decreasing the complexity to $O(\epsilon^{-2}m^2 \log n)$. Both algorithms are mainly based on sequences of shortest path computations. In a previous paper [BCL$^+$03], we gave an improvement on Fleischer's algorithm using dynamic shortest paths algorithms.

In the following we propose an incremental strategy to improve on the running time of these algorithms. We then investigate on our approach through practical tests on different flow networks. We first present the algorithm that we improve.

1 Combinatorial Multicommodity Flow Approximation

The $(1 + \epsilon)$-approximation algorithm for fractional multicommodity flow [BCL$^+$03] is based on a combinatorial understanding of the dual of the multicommodity flow *edge-path* linear program. This dual builds a length function l on the edges of the graph such that the length of an edge is related to the amount of flow it bears.

The process starts by assigning to each edge the same initial length $\delta > 0$, a constant which depends on the parameters of the algorithm, including the approximation factor ϵ, and corresponds to a null amount of flow. Then, the algorithm pushes flow iteratively along single source shortest paths (SSSP) for each commodity while the length of each edge of the paths is increased. The loop ends when every shortest paths from the source of a commodity to its destination are of length more than 1. At this step, the analysis shows that the flow scaled by $log_{1+\epsilon}\left(\frac{1+\epsilon}{\delta}\right)$ fits the capacities, and that the worst case complexity is $O\left(\frac{m \log n}{\epsilon^2}(m + n \log n)\right)$. One may notice that this complexity is the same as the one given by Fleischer [Fle00], even though the use of dynamic shortest paths algorithms yields an improvement of the running time. As a matter of fact, the speed up provided by such algorithms cannot be precisely evaluated unless the structure of the flow network is known.

In the following, we try to cope with the ϵ^{-2} factor in the complexity.

2 An Incremental Approach

In order to ensure a $(1 + \epsilon)$-approximation, each iteration of the previous algorithm pushes a very small amount of flow. We propose an incremental approach based on the idea that the precision is only required at the end of the process. Therefore, our algorithm, reported as Algorithm 1, starts by running the previous algorithm with a large ϵ_0, which pushes a lot of flow quickly. Instead of scaling the flow to fit the capacities, Algorithm 1 scales it with the length function so that it looks like an intermediary step of previous algorithm with a smaller $\epsilon_1 = \epsilon_0/2$. Starting from this intermediary step, the process in finished with ϵ_1, scaled again for $\epsilon_2 = \epsilon_1/2$ and so forth, until Algorithm 1 reaches the required ϵ_f.

Algorithm 1 Incremental multicommodity flow $(1+\epsilon)$-approximation

Require: $G=(V,E)$, $|V|=n$, $|E|=m$, $C=\{(s_i,t_i),\ i=1\dots k\}\subseteq V^2, \epsilon>0$
Ensure: l length function over E s.t. every path $s_i \to t_i$ has length > 1.
Ensure: f $(1+\epsilon)$-approximation of max. multicommodity flow for C on G

1: {Initializing} $t=4$, $\epsilon(t)=2^{-t}$, $\delta(t)=e^{\epsilon(t)}(e^{\epsilon(t)}n)^{-\frac{1}{(\ )}}$, $B(t)=log_{e\ (\ )}(\frac{e\ (\ )}{\delta(t)})$, $\forall e\in E,\ l(e)=\delta(t),\ f_i(e)=0$
2: **while** $\epsilon(t)\geq\epsilon_0$ **do**
3: {Lower bound on the length of a SSSP } $\lambda=\min_{e\in E} l(e)$
4: **while** $\lambda\leq 1$ **do**
5: **for all** $i=1\dots k$ **do**
6: $P\leftarrow$SSSP $(s_i\to t_i)$
7: **while** $l(P)\leq e^{\epsilon(t)}\lambda$ **do**
8: $c_m\leftarrow\min_{e\in P} c(e)$
9: $\forall e\in P,\ f_i(e)\leftarrow f_i(e)+c_m,\ l(e)\leftarrow l(e)e^{\epsilon(t)\frac{}{(\ )}}$
10: $P\leftarrow$USSSP $(s_i\to t_i)$
11: $\lambda\leftarrow\lambda e^{\epsilon(t)}$
12: $t\leftarrow t+1$
13: $\forall e\in E,\ l(e)\leftarrow l(e)\frac{\delta(t)}{\delta(t-1)}e^{\epsilon(t)\frac{(\ )}{(\ )}(\frac{(\ )}{(\ -1)}-2)B(t-1)}$
14: $\forall e\in E,\ \forall i=1\dots k,\ f_i(e)=\frac{B(t)}{B(t-1)}f_i(e)$
15: $\forall i=1\dots k, \forall e\in E,\ f_i(e)=\frac{f\ (e)}{B(t)}$

3 Experimental Results

The analysis of the complexity and approximation performances of Algorithm 1 is a work in progress. Nevertheless, experimental results are encouraging. We report on a set of experiments achieved on variable size multicommodity flow problems, arising in the setting of routing and wavelength assignment for wavelength division multiplexing optical networks [CR02]. The results are depicted as Figure 1.

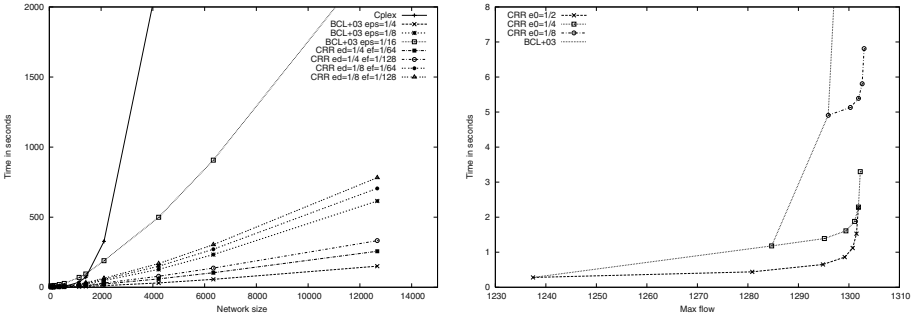


Fig. 1. Running times function of network size (left) or quantity of flow (right).

The left hand side of Figure 1 presents the running time of the optimal resolution of the flow with CPLEX, the algorithm of [BCL+03] with ϵ varying from 1/4 to 1/16, and Algorithm 1 (named CRR) with $\epsilon_0 = 1/4$ or 1/8 and $\epsilon_f = 1/64$ or 1/128. One can see that both combinatorial algorithms are much faster than CPLEX when the size of the problem increases. These results also show that the running time of Algorithm 1 for a given ϵ_0 equals to the running time of Algorithm [BCL+03] for $\epsilon = \epsilon_0$ plus something small and linear in ϵ_f^{-1}.

The right hand side presents the time required by the different combinatorial algorithms to produce a given approximation of the maximum multicommodity flow. These experiments have been achieved on the network of size 500, but the same results are observed with any size. As far as Algorithm 1 for $\epsilon_0 = \epsilon_f$ is exactly Algorithm [BCL+03] for the same parameter, we obtain this *comb graph* like results. One can then realize that Algorithm 1 is able to produce approximated solution much faster than Algorithm [BCL+03]. Moreover, the influence of ϵ_0 on the quality of the solutions of Algorithm 1 is weak, but very strong on the running time.

4 Conclusion

In this paper, we have proposed a combinatorial incremental $(1 + \epsilon)$-approximation algorithm for the fractional multicommodity flow problem, improving on several algorithms of the literature. Practical experiments yield some conjectures on its complexity and approximation performances. A formal analysis of the algorithm is an ongoing work, as well as its integration in an integral multicommodity flow approximation algorithm and further improvements.

References

[BCL+03] M. Bouklit, D. Coudert, J.-F. Lalande, C. Paul, and H. Rivano. Approximate multicommodity flow for wdm networks design. In *Sirocco 10*, Umea, Sweden, june 2003.

[BT97] D. Bertimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[CR02] D. Coudert and H. Rivano. Lightpath assignment for multifibers WDM optical networks with wavelength translators. In *IEEE Globecom'02*, Taiwan, November 2002. OPNT-01-5.

[FF62] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[Fle00] L. Fleischer. Approximating fractional multicommodity flows independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.

[GK98] N. Garg and J. Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.

[Rag94] P. Raghavan. Probabilistic construction of deterministic algorithm: Approximating packing integer programs. *Journal of Computer and Systems Sciences*, 38:683–707, 1994.

Disk Graphs: A Short Survey*

Aleksei V. Fishkin

University of Kiel
Olshausenstr. 40, D-24118 Kiel, Germany
avf@informatik.uni-kiel.de

In general, we define a disk graph (DG) as the intersection graph of a set of disks in the Euclidean plane. Recently, there has been increasing interest in studying the class of DGs. This primary motivated by its applications which can be found in radio networks, map labeling, and in sensor networks, just to name a few. From another side, DGs have a very simple geometric structure. This motivates the study of theoretical problems. Here we give a short survey on DGs. We briefly discuss coloring, independent set and clique. We include hardness results, main ideas used in approximation and online algorithms, lower and upper bounds. We also mention some open questions.

Disk Graphs. There are a number of variants for a DG to be represented by a set of disks. If disks are not allowed to overlap but touch, we have a coin graph (CG). If all disk have unit diameter, we have a unit disk graph (UDG). If the diameter ratio of disks is bounded by some constant σ, we have a σ-disk graph (σ-DG). For an illustration see Fig. 1. As pointed in [5], UDGs need not

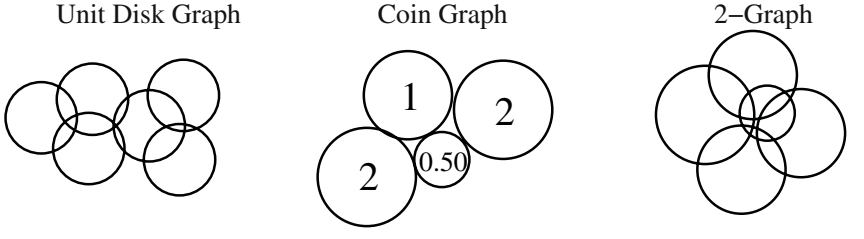


Fig. 1. Disk graphs

to be perfect since any odd cycle of length five or grater is a UDG. Similarly, UDGs need not be planar; in particular, any clique of size five or more is a UDG. From another side, every planar graph is a CG [15]. Thus many of the known efficient algorithms for perfect and planar graphs do not apply to (unit, σ) DG. Unfortunately, the recognition problem of a (unit, σ) DG is also NP-hard [4, 12]. So, we always need to distinguish whether an algorithm uses disks as the input or not.

* Partially supported by EU-Project CRESCCO, IST-2001-33135, and by EU-Project ARACNE, HPRN-CT-199-00112.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 260–264, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Independent Set. An independent set (IS) of a graph $G = (V, E)$ is a vertex subset $V' \subseteq V$ such that for any pair of vertexes in V', there is no edge between them in E. The *independence number* $\alpha(G)$ is defined as the size of a maximal independent set (MIS) in G.

It is NP-hard to find an MIS in a UDG, even if its disks are given [5, 21]. However, the neighborhood structure of a unit disk in the plane gives two nice ideas, see Fig 2. First, there are at most 5 non-intersecting unit disks around. Second, 5 reduces to 3, if the unit disk has the lowest y-coordinate amoung all disks. Accordingly, we can use GREEDY: Take disks one by one, put a disk into

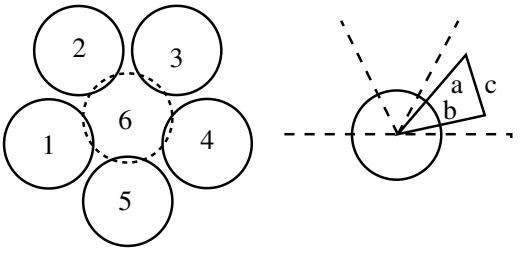


Fig. 2. The neighborhood structure

an IS if it does not intersect any disk in the IS. (Of course, GREEDY not need to use disks.) In the worst case, any disk in the IS blocks at most 5 other disks. So, the size of IS is at least $\alpha(G)/5$ [10, 16]. Thus GREEDY is a 5-competitive algorithm. From Fig 2, no deterministic online algorithm can be better than 5-competitive.

In the offline case, the approximation ratio of GREEDY can be improved from 5 to 3 [17]. We work in the same way but take disks in the order of non-decreasing y-coordinates. Here we also not need to use unit disks. We can always find a vertex whose neighborhood does not contain an IS of size larger than 3. (A simple enumeration completes in $O(|V|^5)$ time.) We add such a vertex to an IS, remove all its neighbors, and repeat the procedure until the input graph is empty. So, either disks are used or not, there is a 3-approximation algorithm, but the running time differs.

Regarding disk graphs, we just follow old ideas. In the online case, we use GREEDY. Either disks are given or not (some simple arguments), GREEDY is an optimal $(|V| - 1)$-competitive algorithm [6]. In the offline case, we can first order disks by non-decreasing size, and then use GREEDY on disks in this order. So, we get a 5-approximation algorithm. As before, we also not need to use disks. It suffices to identify a vertex whose neighborhood does not contain an IS of size at most 6. This also gives a 5-approximation algorithm [17].

Finally, given a set of (unit) disks in the plane, we can find an IS of size at least $(1 - \varepsilon)\alpha(G)$, for any fixed $\varepsilon > 0$ [14, 18, 8]. In other words, there exists a polynomial time approximation scheme, or a PTAS for short. To be able to cope

with the problem, one uses the ideas of the shifting technique [2, 13]. Due to the space limitations we omit it.

Clique. A clique of a graph $G = (V, E)$ is a vertex subset $V' \subseteq V$ such that such that for any pair of vertexes in V', there is an edge between them in E. The *clique number* $\omega(G)$ is defined as the size of a maximal clique (MC) in G.

Surprisingly, it turns out that an MC in an UDG can be found in polynomial time. Provided unit disks in the plane, the main idea is rather simple [5]. Consider two unit disks x and y with centers p_x and p_y, respectively. Obviously, they intersect iff the distance $d(p_x, p_y) \leq 1$. Then, let $A_{x,y}$ be the set of all unit disks whose centers are in both (closed) circles around p_x and p_y with radius $d(p_x, p_y)$, see Fig 3. Certainly every disk in $A_{x,y}$ must intersect both x and y. $A_{x,y}$ not need to induce a clique, but it must induce complements of a bipartite graph. This can be seen by drawing the line between p_x and p_y. The disks in either half intersect pairwise. An MC in complements of a bipartite graph, i.e. an MIS in a bipartite graph, can be found in time $O(|V|^{2.5})$, by matching. Any clique must be the form of $A_{x,y}$. So, we can find an MC by looking at all $A_{x,y}$. The

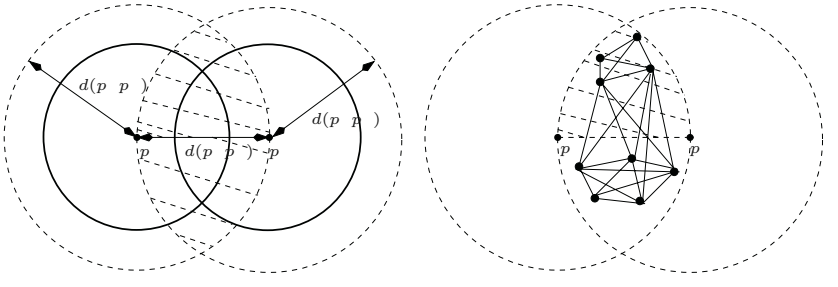


Fig. 3. Two disks

running time of the algorithm is $O(|V|^{4.5})$, but improves to $O(|V|^{3.5})$ [3]. Later, in [20], these nice ideas were adopted so as to remove the dependence on unit disks. First, a greedy algorithm is used to find a cobipartite neighborhood edge elimination ordering. That is, an ordering $e_1, e_2, \ldots, e_{|E|}$ such that for each edge $e_i = (x, y)$ the set of vertices $N_i \subseteq V$ which are adjacent to both x and y in the subgraph induced by $\{e_i, e_{i+1}, \ldots, e_{|E|}\}$, induces a cobipartite subgraph in G. As one can see, if G is an UDG then such an ordering exists. Again, by matching we can find MCs for all N_i. An MC of G must be among them. The resulted algorithm is also a robust algorithm. Given a graph, it either finds an MC or outputs that the given graph is not an UDG.

Unfortunately, there are no results for general DGs. The complexity of the problem is also open.

Coloring. A coloring of a graph $G = (V, E)$ is an assignment of colors (positive integers) to each vertex in V such that no edge in E connects two identically colored vertices. The *chromatic number* $\chi(G)$ is the defined as the minimal

number of colors in a coloring of G. Clearly, we need at least $\omega(G)$ colors, i.e. $\omega(G) \leq \chi(G)$.

It is NP-complete to decide whether a UDG can be colored with 3 colors, even if its disks are given [5]. (So, there is no 4/3-approximation algorithm unless P = NP.) Furthermore, deciding k-colorability remains NP-complete, for any fixed k [10].

For a UDG, we simply recall the neighborhood structure of a unit disk v in V. Clearly, there are at most $5(\omega(G) - 1)$ neighbors of v. So, we can use First-Fit: Take vertices one by one, and assign a least available color to the vertex. Then, First-Fit uses at most $5\omega(G) + 4$ colors [6, 7]. This gives a 5-competitive algorithm. Regarding a lower bound, no algorithm can be better than 2-competitive [9]. Later, this bound was improved to 4 [22].

Indeed, we can improve the performance of First-Fit in the offline case by processing the disks in the plane in the order of non-increasing y-coordinates. Similarly, we can show that First-Fit uses at most $3\omega(G) - 2$ colors [19]. This gives a 3-approximation algorithm. We can also adopt the algorithm to the case when no geometric information about disks is known. We fist recursively order the vertices by non-increasing degree, and then, apply First-Fit by tracking vertices back. Again, such the smallest-degree-last First-Fit algorithm uses at most $3\omega(G) - 2$ colors [10]. So, it achieves approximation ratio 3.

Interestingly, the smallest-degree-last First-Fit also performs well on general DGs. Its approximation ratio can be proven to be 5 [16]. Furthermore, by exploring the neighborhood structure of a disk in the plane one can color any DG with at most $6\omega(G) - 6$ colors.

In the online case, no constant competitive algorithm can be found. As it was shown in [7, 11], $\Omega(\log |V|)$ is a lower bound on the competitive ratio of any online algorithm, even if it uses disks as the input. From another side, First-Fit is $\log |V|$-competitive [6], that gives an optimal online algorithm.

References

1. Agarwal, P. K., Overmars, M., and Sharir, M. Computing maximally separated sets in the plane and independent sets in the intersection graph of unit disks. In *Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms* (2004). To appear.
2. Baker, B. S. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM 41* (1994), 153–180.
3. Breu, H. *Algorithmic Aspects of constrained unit disk graphs.* PhD thesis, University of british Columbia, 1996.
4. Breu, H., and Kirkpatrick, D. G. Unit disc graph recognition is NP-hard. *Computational Geometry: Theory and Applications 9* (1998), 3–24.
5. Clark, B. N., Colbourn, C. J., and Johnson, D. S. Unit disk graphs. *Discrete Mathematics 86* (1990), 165–177.
6. Erlebach, T., and Fiala, J. Independence and coloring problems on intersection graphs of disks. manuscript, 2001.
7. Erlebach, T., and Fiala, J. On-line coloring of geometric intersection graphs. *Computational Geometry: Theory and Applications 23*, 2 (2002), 243–255.

8. Erlebach, T., Jansen, K., and Seidel, E. Polynomial-time approximation schemes for geometric graphs. In *Proceedings the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA'01)* (Washington, DC, 7–9 2001), pp. 671–679.
9. Fiala, J., Fishkin, A. V., and Fomin, F. Off-line and on-line distance constrained labeling of disk graphs. In *Proceedings 9th Annual European Symposium on Algorithms* (Arhus, 2001), LNCS 2161, pp. 464–475.
10. Gräf, A., Stumpf, M., and Weisenfels, G. On coloring unit disk graphs. *Algorithmica 20*, 3 (1998), 277–293.
11. Gyárfás, A., and Lehel, J. On-line and first fit colorings of graphs. *Journal of Graph Theory 12* (1988), 217–227.
12. Hliněný, P., and Kratochvíl, J. Representing graphs by disks and balls. *Discrete Mathematics 229* (2001), 101–124.
13. Hochbaum, D., and Maass, W. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the ACM 32* (1985), 130–136.
14. HuntIII, H. B., Marathe, M. V., Radhakrishnan, V., Ravi, S. S., Rosenkrantz, D. J., and Stearns, R. E. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *J. Algorithms 26*, 2 (1998), 238–274.
15. Koebe, P. Kontaktprobleme der konformen Abbildung. In *Math.-Phys. Klasse*, vol. 88. Berichte Verhande. Saechs. Akad. Wiss. Leipzig, 1936, pp. 141–164.
16. Malesińska, E. *Graph theoretical models for frequency assignment problems*. PhD thesis, Technical University of Berlin, Germany, 1997.
17. Marathe, M. V., Breu, H., Hunt III, H. B., Ravi, S. S., and Rosenkrantz, D. J. Simple heuristics for unit disk graphs. *Networks 25* (1995), 59–68.
18. Matsui, T. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *Proceedings The Japan Conference on Discrete and Computational Geometry* (1998), LNCS 1763, Springer Verlag, pp. 194–200.
19. Peeters, R. On coloring j-unit sphere graphs. Tech. rep., Department of Economics, Tilburg University, 1991.
20. Raghavan, V., and Spinrad, J. Robust algorithms for restricted domains. *Journal of Algorithms 48(1)* (2003), 160–172.
21. Wang, D., and Kuo, Y.-S. A study on two geometric location problems. *Information Processing Letters 28* (1988), 281–286.
22. Y.-T. Tsai, Y.-L. Lin, F. H. The online firstfit algorithm for radio frequency assignment problems. *Information Processing Letters 84*, 4 (2002), 195–199.

Combinatorial Techniques for Memory Power State Scheduling in Energy-Constrained Systems

Claude Tadonki[1,⋆], Mitali Singh[2], Jose Rolim[1], and Viktor K. Prasanna[2]

[1] Dept. of Theoretical CS, Informatics Center, University of Geneva
Avenue du general dufour 24, Geneva, Switzerland
claude.tadonki@unige.ch, rolim.jose@cui.unige.ch
[2] Dept. of Computer Science, University of Southern California
Los Angeles, CA-90089, USA
{mitali,prasanna}@usc.edu

Abstract. Energy has emerged as a critical constraint for a large number of portable, wireless devices. For data intensive applications, a significant amount of energy is dissipated in the memory. Advanced memory architectures support multiple power states of memory banks, which can be exploited to reduce energy dissipation in the system. We present a general methodology using combinatorial graph scheduling techniques, which can be used for obtaining efficient memory power management schedules for algorithms. Additional techniques like tiling further improve the efficiency of our approach. Our simulation results show that we can obtain over 98% energy reduction in the memory energy for the Transitive Closure using our methodology.

Keywords: Algorithm, energy, complexity, graph, memory, schedule.

1 Introduction

Due to the growing popularity of embedded systems and related[1–3], energy has emerged as a new optimization metric for system design. As the power availability in most of these systems is limited by the battery power of the device, it is critical to reduce energy dissipation in these systems to maximize their operation cycle.

The topic of energy reduction has been intensively studied in the literature and is being investigated at all levels of system abstraction, from the physical layout to software design. The research at the architecture level has led to innovations such as the dynamic voltage and frequency scaling of the processor [8] and in design of power efficient memory systems. The latter provides support for dynamic transition of memory banks into various power states and partial activation of rows or columns in memory banks [4]. In order to design energy efficient applications it is required that the architectural power management features are exploited at higher levels. Till now, most researchers have focused upon the compiler[5–7] and the operation system level, where the target architecture

⋆ Supported by CRESCCO Project and FNRS Funds.

R. Solis-Oba and K. Jansen (Eds.): WAOA 2003, LNCS 2909, pp. 265–268, 2004.
© Springer-Verlag Berlin Heidelberg 2004

is taken into account, and the optimization techniques are specific to the chosen application. Some contributions have also been proposed at the algorithmic level. The goal is to optimize an algorithm for energy dissipation in the memory and other devices[9–11]. The solutions proposed are again specific to the chosen algorithms.

In this paper, we provide a combinatorial method to schedule memory power state in an efficient way.

2 A Model of Energy Evaluation

We assume that the energy spent for running an algorithm mainly depends on :

- the operations performed by the processor (arithmetic and logical operations, comparisons, etc...);
- the operations performed on the memory (read/write operations, storage, and state switchings);
- the operations performed between the processor and the memory (transfers between the global memory and the cache).

Given an algorithm running on an entry of size n, if $T(n)$, $S(n)$, and $C(n)$ denote respectively the time complexity (number of processors operations), the space complexity (the amount of memory used), and cache complexity (number of cache missing), the energy complexity of the algorithm is approximated by

$$E(n) = \tau T(n) + \Omega Q^T S(n) + \Omega W^T C(n). \tag{1}$$

3 Our Methodology

In this section, we describe our methodology to obtain efficient power state schedules for the memory system.

3.1 Timing Function

Given a tasks graph $G = (X, D)$, we present a method to assign each node a valid execution time. We assume that each task takes one unit time (a time cycle). First, we consider an ordinal correspondence $\eta : X \to \{1, \cdots, |X|\}$ (thus a total order in X). Next, we associate to each node $x \in X$ a value $\pi(x)$ initialized with its *in-degree* given by

$$|\{s \in X : (s, x) \in D\}|. \tag{2}$$

For any subset $W \subseteq X$, the unique node x such that $\pi(x) = 0$ and $\eta(x)$ is minimal is denoted by $\chi(W)$. Hence, the following algorithm provides an execution time of each nodes in X.

Algorithm 1. Algorithm for tasks execution times

```
for x ∈ X
  π(x) ← |{s ∈ X : (s, x) ∈ D}|
endfor
S ← X
T ← |S|
for t ← 1 to T
  x ← χ(S)
  π(x) ← t
  for y ∈ D(x)
    π(y) ← π(y) − 1
  endfor
  S ← S − {x}
endfor
```

3.2 Memory Power State Management

Consider the previous graph model $G = (X, D)$ and a given timing function π. For a given node $x \in X$, since each node of $D(x)$ is executed at a different time, we can write the set $D(x)$ in an execution time ordered form $\{x_1, x_2, \cdots, x_{c(x)}\}$, where $c(x) = |D(x)|$ (the out-degree of x). Next, we consider the set $\{\delta_1(x), \delta_2(x), \cdots, \delta_{c(x)-1}\}$, where

$$\delta_j(x) = \pi(x_{j+1}) - \pi(x_j). \tag{3}$$

Now, consider that the set of memory state is $\mathcal{S} = \{0, 1, \cdots, m\}$, where 0 stands for the *active state*, and corresponding activation cost and storage cost are $W = (w_0, w_1, w_2, \cdots, w_m)$ $(w_0 = 0)$ and $Q = (q_0, q_1, \cdots, q_m)$ respectively. We assume that $w_s < w_{s+1}$, $s = 0, ..., m-1$. We emphasize on the fact that 0 stand for active state and m stand for inactive state.
For a given $j \in \{1, \cdots, c(x) - 1\}$, we consider

$$\sigma(j) = \min\{s, 0 \le s \le m : \delta_j(x) q_s > w_s\}. \tag{4}$$

Since $w_0 = 0$, $\sigma(j)$ is well defined for each j, and it indicates the best memory state for x in the time interval between the computation of x_j and that of x_{j+1}.

4 Simulation Results

Lastly, we present our simulation results. The simulations were performed using the simulator described in [9]. Table 1 shows our results for Transitive Closure, where a gain of 98% was obtained.

5 Conclusion

We have proposed a combinatorial method to schedule tasks graph with an emphasize on energy dissipation. Although the method is applied on a graph model,

Table 1. Simulation Results with the Transitive Closure

n	T	$\tilde{T}$	$\frac{\tilde{T}-T}{T}$	E	$\tilde{E}$	$\frac{E-\tilde{E}}{E}$
32	0.012	0.12	0.012	0.052	0.0006	0.987
64	0.090	0.092	0.014	0.399	0.0056	0.985
128	0.723	0.738	0.020	3.143	0.0562	0.982
256	5.830	5.954	0.021	25.308	0.4644	0.981

it can be also used with recurrence equations specifications through appropriate syntaxical manipulations. Simulation results show that the method yields significant energy reduction. We think that the study of a dynamic version of the method would be interesting. Tiling issue should be analyzed in more details, since it is the key for successful optimization through data/instructions remapping.

References

1. W. Wolf. Software-Hardware Codesign of Embedded Systems. *Proceedings of the IEEE* , volume 82, 1998.
2. R. Ernst. Codesign of Embedded Systems: Status and Trends. *IEEE Design and Test of Computers* , volume 15, 1998.
3. Manfred Schlett. Trends in Embedded Microprocessors Design. *IEEE Computer*, 1998.
4. "Mobile SDRAM Power Saving Features," Technical Note TN-48-10, MICRON, http://www.micron.com
5. W. Tang, A. V. Veidenbaum, and R. Gupta. Architectural Adaptation for Power and Performance. *International Conference on ASIC*, 2001.
6. L. Bebini and G. De Micheli. Sytem-Level Optimization: Techniques and Tools. *ACM Transaction on Design Automation of Electronic Systems*, 2000.
7. T. Okuma, T. Ishihara, H. Yasuura. Software Energy Reduction Techniques for Variable-Voltage Processors. *IEEE Design and Test of Computers* , 2001.
8. J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic Voltage Scaling on a Low-Power Microprocessor," UbiCom-Tech. Report, 2000.
9. M. Singh and V. K. Prasanna. Algorithmic Techniques for Memory Energy Reduction. *Worshop on Experimetal Algorithms*, 2003.
10. S. Sen and S. Chatterjee. Towards a Theory of Cache-Efficient Algorithms. *SODA*, 2000.
11. D.F. Bacon, S.L. Graham, and O.J. sharp. Compiler Transformations for High-Performance Computing. *Hermes*, 1994.
12. C. Tadonki. A Recursive Method for Graph Scheduling. *Interantional Symposium on Paralle and Distributed Computing*, July 2002.

Author Index